Advant® OCS
with Master software

# Bulk Data Handling and Reuse

## Reference Manual

3BDS100581R1

## Use of **DANGER**, **WARNING**, **CAUTION**, and **NOTE**

This publication includes, **DANGER**, **WARNING**, **CAUTION**, and **NOTE** information where appropriate to point out safety related or other important information.

| | |
|---|---|
| **DANGER** | Hazards which could result in severe personal injury or death |
| **WARNING** | Hazards which could result in personal injury |
| **CAUTION** | Hazards which could result in equipment or property damage |
| **NOTE** | Alerts user to pertinent facts and conditions |

Although **DANGER** and **WARNING** hazards are related to personal injury, and **CAUTION** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all **DANGER**, **WARNING**, and **CAUTION** notices.

## TRADEMARKS

Advant, AdvaCommand, AdvaInform, and AdvaBuild are registered trademarks of ABB Asea Brown Boveri Ltd., Switzerland.

## NOTICE

The information in this document is subject to change without notice and should not be construed as a commitment by ABB Automation Products GmbH. ABB Automation Products GmbH assumes no responsibility for any errors that may appear in this document.

In no event shall ABB Automation Products GmbH be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB Automation Products GmbH be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without ABB Automation Products GmbH’s written permission, and the contents thereof must not be imparted to a third party nor be used for any unauthorized purpose.

The software described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

Copyright © ABB Automation Products GmbH 1999

# TABLE OF CONTENTS

## Chapter 1 - Introduction

1.1 Before You Begin ........ 1-1
1.2 Equipment Requirements ........ 1-1
1.3 How to Use This Book ........ 1-1
1.4 Conventions ........ 1-2
1.5 Related Documentation ........ 1-3
1.6 Release History ........ 1-3
1.7 Terminology ........ 1-3

## Chapter 2 - Import

2.1 Import Process ........ 2-1
2.2 Instantiate Objects ........ 2-1
2.3 Instantiate Aspects ........ 2-4
2.4 Create Objects and Aspects ........ 2-5
2.5 Create Configuration Data ........ 2-6
2.6 Starting the Import Function ........ 2-6
2.7 Import Progress ........ 2-10
2.8 Error Logging ........ 2-10

## Chapter 3 - Export

3.1 Export Process ........ 3-1
3.2 Starting the Export Function ........ 3-2
3.3 Processing ........ 3-5
3.4 Output Data ........ 3-6
3.5 Export Logging ........ 3-8
3.6 Set-Files ........ 3-9
3.7 Working with Unique Names ........ 3-9

## Chapter 4 - Reference - Import File Formats

4.1 Instantiate Signal Objects ........ 4-1
4.2 Instantiate Station/Fieldbus Objects ........ 4-1
4.3 Instantiate Process Objects ........ 4-2
4.4 Instantiate Board/Module Objects ........ 4-3
4.5 Instantiate CAEE Signal Objects ........ 4-3
4.6 Instantiate CAEE Process Objects ........ 4-4
4.7 Instantiate CAEE Board Objects ........ 4-5
4.8 Instantiate Aspects ........ 4-6

# CONTENTS (continued)

4.9 Create Aspects and Objects....................4-7
  4.9.1 Create Advant Objects....................4-7
  4.9.2 Create Structure Aspects....................4-9
  4.9.3 Create Basic Property Aspects....................4-10
  4.9.4 Create Document Aspects....................4-10
  4.9.5 Create Parameter Aspects....................4-12
  4.9.6 Create Node Control Software Aspects....................4-13
  4.9.7 Create Circuit Aspects....................4-14
  4.9.8 Create Type Circuit Aspects....................4-15
  4.9.9 Create DBSection Aspects....................4-16
  4.9.10 Create DBElement Aspects....................4-17
4.10 Create Electrical Diagram Aspects....................4-18
4.11 Create Object Types....................4-19
4.12 Create Structures....................4-20
4.13 Create Structure Display Settings....................4-21
4.14 Create Document Templates....................4-22
4.15 Create Parameter Aspect Types....................4-23
4.16 Create Document Objects....................4-23
4.17 Rename Aspects....................4-25
4.18 Rename Objects....................4-25

## Chapter 5 - Maintenance

5.1 Error Messages....................5-1
5.2 Recommendations....................5-5
  5.2.1 File Format of Import Data Files....................5-5

## Appendix A - Overviews, Samples

A.1 Overview - Instantiate Behavior of Structure Aspects....................A-1
A.2 Sample - Instantiate Signal Objects....................A-3

## INDEX

# ILLUSTRATIONS

Figure 2-1. Structure Location with Three Parent Objects ........ 2-2
Figure 2-2. Instantiate Substructure Objects ........ 2-3
Figure 2-3. Import Data Window ........ 2-6
Figure 2-4. Add Source Files Window ........ 2-8
Figure 2-5. Override Options Window ........ 2-9
Figure 2-6. Progress Window ........ 2-10
Figure 2-7. Log File Sections - Principle ........ 2-11
Figure 3-1. Export Window ........ 3-3
Figure 3-2. Folder Select Window ........ 3-4
Figure 3-3. Export Progress Window ........ 3-5
Figure 3-4. Export Output Data (Sample) ........ 3-6
Figure 3-5. Export Log file (Sample) ........ 3-8
Figure 3-6. Import - Rename Files ........ 3-10
Figure 5-1. Example Import Data File ........ 5-5
Figure A-1. Examples - Naming Rules ........ A-3
Figure A-2. Signal Data File ........ A-3
Figure A-3. Example - Instantiate Signal Object ........ A-4

# TABLES

Table 1-1. Related Documents ........ 1-3
Table 3-1. Overview Export Files ........ 3-7
Table 4-1. File Format of Instantiate Signal Objects ........ 4-1
Table 4-2. File Format of Instantiate Station Objects ........ 4-1
Table 4-3. File Format of Instantiate Process Objects ........ 4-2
Table 4-4. File Format of Instantiate Board/Module Objects ........ 4-3
Table 4-5. File Format of Instantiate CAEE Signal Objects ........ 4-3
Table 4-6. File Format of Instantiate CAEE Process Objects ........ 4-4
Table 4-7. File Format of Instantiate CAEE Board Objects ........ 4-5
Table 4-8. File Format of Instantiate Aspects ........ 4-6
Table 4-9. File Format of Create Aspects and Objects ........ 4-7
Table 4-10. Attribute Definitions - Create Advant Objects (Name) ........ 4-8
Table 4-11. Attribute Definitions - Create Advant Objects (ARD) ........ 4-8
Table 4-12. Attribute Definitions - Create Structure Aspects ........ 4-9
Table 4-13. Attribute Definitions - Create Basic Property Aspects ........ 4-10
Table 4-14. Attribute Definitions - Create Document Aspects, Using Templates ........ 4-10

# TABLES (continued)

Table 4-15. Attribute Definitions - Create Document Aspects, Using External File..... 4-11
Table 4-16. Attribute Definitions - Create Parameter Aspects ..................................... 4-12
Table 4-17. Attribute Definitions - Create Node Control Software Aspects ................ 4-13
Table 4-18. Attribute Definitions - Create Circuit Aspects .......................................... 4-14
Table 4-19. Attribute Definitions - Create Type Circuit Aspects ................................. 4-15
Table 4-20. Attribute Definitions - Create DBSection Aspects.................................... 4-16
Table 4-21. Attribute Definitions - Create DBElement Aspects................................... 4-17
Table 4-22. File Format of Create Electrical Diagram Aspects.................................... 4-18
Table 4-23. File Format of Create Object Types......................................................... 4-19
Table 4-24. Example - File Format of Create Object Types ......................................... 4-20
Table 4-25. File Format of Create Structures............................................................... 4-20
Table 4-26. Example - File Format of Create Structures .............................................. 4-20
Table 4-27. File Format of Create Structure Display Settings...................................... 4-21
Table 4-28. Example - File Format of Create Structure Display Settings .................... 4-21
Table 4-29. File Format of Create Document Templates.............................................. 4-22
Table 4-30. Example - File Format of Create Document Templates ............................ 4-22
Table 4-31. File Format of Create Parameter Types ..................................................... 4-23
Table 4-32. Example - File Format of Create Parameter Types.................................... 4-23
Table 4-33. File Format of Create Document Objects .................................................. 4-24
Table 4-34. Attribute Definitions - Rename Aspects.................................................... 4-25
Table 4-35. Attribute Definitions - Rename Objects .................................................... 4-25
Table 5-1. Error Messages in the Import Log File........................................................ 5-1
Table A-1. Import/Instantiate Behavior of Structure Aspects. ..................................... A-1

# Chapter 1 Introduction

## 1.1 Before You Begin

This manual is intended as a guide for working with the Structure Builder's **Import** and **Export** functions. It is a complete description of both functions and includes a collection of the various import file formats.

Further functions covering **Bulk Data Handling** are:

- Cut/Copy & Paste of objects (also between Projects)
  described in *Structure Builder User's Guide*
- Modify filtered parameter attributes
  described in *Parameter Builder User's Guide*
- Multiple Execution
  described in *Electrical Diagram Builder User's Guide*
- Multiple Selection/Execution (Update Parameter, Generate DB, Initialize, Locate Circuit) described in *AMPL Control Configuration Commands for Structure Builder.*

More information about **Reuse** (creating and working with libraries) give:

- *Engineering Methods User's Guide*
  Describes engineering with AdvaBuild engineering software.
- *Object Library Reference Manual*
  Describes the content of the Object Library.
- *AMPL Control Configuration Commands for Structure Builder*
  Describes configuring Advant objects for libraries in section 4.3.22.

## 1.2 Equipment Requirements

For a description of the system requirements please read the *Product Guide* and the *Release Notes* of the individual builders.

## 1.3 How to Use This Book

This book is organized into following chapters:

- Chapter 1 Introduction
- Chapter 2 Import
- Chapter 3 Export
- Chapter 4 Reference - Import File Formats
- Chapter 5 Maintenance

- Appendix A Overviews, Samples
- Index.

## 1.4 Conventions

The following conventions are used throughout this manual for the presentation of material:

- The words in proper names of screen elements (for example, the title in the title bar of a window, the label for a field of a dialog box) are initially capitalized.
- The names of keyboard keys are boldfaced.
  - Initial capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, the **Return** key or **<Return>**.
  - Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the **space bar**, **comma key**, and so on.
  - In descriptions, the word “key” is used in conjunction with its name. For example, pressing the **Return** key initiates the process.
  - In tables and procedures, angle brackets replace the word “key.” For example, press **<Return>**.
  - A dash is used to indicate that you must simultaneously press several keys. For example, **<Ctrl-C>** indicates that you must hold down the **Control** key while pressing the **C** key.
  - The phrase “press and release” is used to indicate that you sequentially press several keys. For example, to close a window, press and release **<Alt>**, and then **<F4>**. This indicates that you press and release the **Alt** key, and then you press and release the **F4** function key.
- The names of push and toggle buttons are boldfaced.
  - Initial capital letters are used for the name of a push or toggle button labeled on the display. For example, the **OK** button.
  - In descriptions, tables and procedures, the word “button” is used in conjunction with its name. For example, pressing the **OK** button initiates the process.
- The names of menus and menu items are boldfaced.
  - Initial capital letters are used for the name of menus listed in the menu bar. For example, the **File** menu.
  - Initial capital letters are used for the name of menu items. For example, to open a new window, choose **New** from the **File** menu.
- System prompts/messages are shown in the Courier font, and user responses/input are in the boldfaced Courier font. For example, if you enter a value out of range, the following message is displayed:

  ```
  Entered value is not valid. The value must be 0-30.
  ```

  In a tutorial, you may be told to enter the string TIC132 in a field. The string is shown as follows in the procedure:

  **`TIC132`**

## 1.5 Related Documentation

Table 1-1 lists all related documentation.

*Table 1-1. Related Documents*

| Title | Description |
|---|---|
| Structure Builder User's Guide | This book describes the creation of projects, Advant objects, structures and aspects and how to navigate in projects |
| Control Configuration Functions User's Guide | Describes the integration of the Application Builder, Function Chart Builder and On-line Builder into the Structure Builder. |
| Application Builder User's Guide | Contains a description of AMPL application programs configuration. |
| Function Chart Builder User's Guide | Contains a description of AMPL application programs configuration. |
| Parameter Builder User's Guide | Describes how to handle parameters for process objects and for OCS objects. |
| Document Builder User's Guide | Describes the support of handling various types of documents, including the handling of templates |
| Electrical Diagram Builder User's Guide | Describes the electrical plant engineering and the handling of circuit diagrams. |
| Object Library Reference Manual | Describes the content of the Object Library and is delivered with the AdvaBuild Engineering Software. |
| Engineering Methods User's Guide | This book describes the workflow and the principles of object oriented engineering by using AdvaBuild engineering software |

## 1.6 Release History

This is the first product release of the *Bulk Data Handling & Reuse Reference Manual*.

## 1.7 Terminology

### Advant Object

Advant object Technology offers a methodology to model real world objects with Advant objects, aspects, and views. All entities you work with during the engineering process, like main plant equipment, components and devices, information entities like signals, or location entities like areas are referred to as Advant objects. Each Advant object may have several aspects.

### Aspect

There are often several ways to look at an Advant object, depending on how you want to handle it. When for example an Advant object is representing a valve, then you might be interested in a drawing of that valve or in its loop. Then you may create with a corresponding Builder an CAEE aspect for the drawing and a control function aspect for the loop. The aspect stores the corresponding data. An aspect is an instance of an aspect type. Normally a Builder supplies some aspect types it is able to work on. You can start a Builder on an aspect by opening one of its views.

### Aspect System

The Aspect Request Broker communicates with a Builder via a so-called Aspect System Interface. When some Builders share an aspect system interface, these Builders are referred to as an aspect system. Therefore, an aspect system may consist of one or several Builders.

### Aspect Type

Each Builder supplies several aspect types. Examples of aspect types are basic property aspect, structure aspect, or parameter aspect and its sub types or category like BoardParameter, SignalParameter, StationParameter and so an. An aspect type is designed for special tasks. These tasks will be performed by the corresponding Builders or aspect systems. An aspect type works as a template for the creation of aspects.

### Basic Property Aspect Type

The basic property aspect type is one of the aspect types supplied by **Structure Builder**. It holds a name and the type of an Advant object. As an Advant object may have several names, it may have several basic property aspects. You can refer to a specific one with the help of its name category, for example with Customer ID.

An Advant object may have for example a customer ID “Turbine A” and a location ID “Turbine A 2nd floor”.

### Object

See Advant object.

### Parent Object

Advant objects can be visualized in a pool or in a structure window. In a structure Advant objects may have a parent-child relation. A parent object may contain several child objects. In this way you can create a hierarchy of Advant objects.

### Project Global Data Aspect Type

At project creation time a project global data aspect will be created as an aspect of the project object. You cannot delete this aspect nor create another one. The project global data aspect type stores general information about a project.

# Chapter 2 Import

## 2.1 Import Process

The characteristic of the import is to load bulk data into the system. The data are stored in Import Data Files, flat ASCII files with *.csv format.

The data scope is either Advant objects or single aspects, which can be instantiated or created.

Instantiate means that you have two projects (can also be combined in one). One is a reference (called library or source project), which keeps well prepared Advant objects and their different aspects. The other is the target project where the objects of the library project have to be copied/instantiated into. The data file describes, which library objects have to be instantiated and which default data have to be substituted by actual values.

Create means that you can load bulk data without referencing any library objects and aspects. In this case all necessary data has to be brought in from the Import Data File.

## 2.2 Instantiate Objects

Instantiate an Advant object is creating a new object by copy an existing object with all its aspects from the library project.

Depending on the aspect type, which is copied during the instantiating several data have to be provided. They have to be stored in a comma separated file (*.csv file), where one row describes the data for one object to instantiate.

There are some basic mechanism how the import can run and handle the different aspect types. They are described in this section. Which of this basic mechanism are used and how the import data file has to be interpreted is predefined for some file formats (see Chapter 4, Reference - Import File Formats).

1. **Library reference**

   The library reference identifies the object in the library project, from which the new object should be instantiated. The value must fit to a basic property aspect (category **Customer ID**, attribute **Name**).

   If the defined object cannot be found in the library project, an error message is written into the Log file.

   It can be more than one library reference defined.

2. **Advant object Name**

   This will be the name for the instantiated object. When instantiate an aspect of type basic property (category **Customer ID**) the Name attribute is set to this value.

   If the value is empty, no instantiate takes place.

3. **Description**

   When instantiate an aspect of type basic property (category Customer ID) the Description attribute of the instantiated aspect is set to this value.

4. **Structure Parent Objects**

   When instantiate an aspect of type structure a target location in the structure has to be given. For this an existing object (the parent object) has to be defined where the instantiated object has to be placed below.This has to be done for each structure.
   If a defined parent object cannot be located when import runs, the structure aspect will not be instantiated. A warning message is put into the Log file.
   Two different parent object definitions are supported:

   a. The parent object is identified by its name. The value must fit to a basic property aspect (category **Customer ID**, attribute **Name**).

   b. Three parent objects define the location place. The first object is identified by its name (same as above at a.). The 2nd and the 3rd object are identified by there relative reference designation. Figure 2-1 shows an example where the 1st object is named `Node1`, the 2nd object has the relative reference designation `DI5` in the OCS structure and the 3rd object has `1` as relative reference designation. The object `Sig1` will be placed below the 3rd parent object.

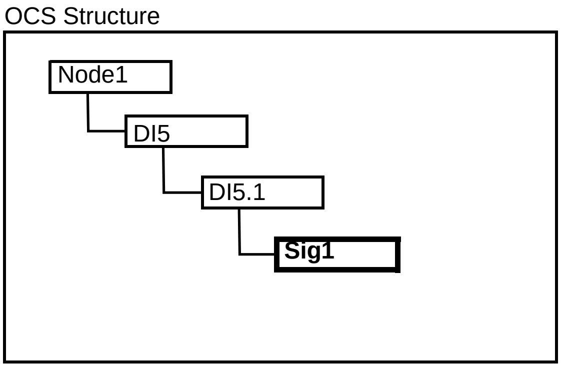


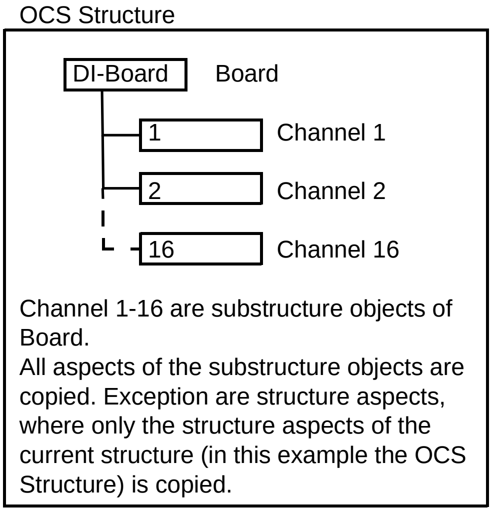
*Figure 2-1. Structure Location with Three Parent Objects*

5. **Instantiate with substructure objects**

   When instantiate an aspect of type structure it is possible to instantiate also substructure objects, if the library object has substructure objects.

   The substructure objects will than be created in the target project, too.

   Figure 2-2 shows an example of a library object with substructure objects in the OCS structure. When instantiating from the DI-Board object, the channel objects will be created, too.



Channel 1-16 are substructure objects of Board.
All aspects of the substructure objects are copied. Exception are structure aspects, where only the structure aspects of the current structure (in this example the OCS Structure) is copied.

*Figure 2-2. Instantiate Substructure Objects*

6. **Relative Reference Designation**

   When instantiate an aspect of type structure the relative reference designation can be set to this value.

7. **Parameter attributes**

   When instantiate an aspect of type parameter, all supplied attributes, which fit to the parameter aspect definition are set to the actual value.

### Identification of Objects and Aspects

During import Advant objects (source, target objects and parent objects in structures) are identified by their name. Alternatively parent objects in structures can be identified by the *absolute reference designation*, which requires a separate mapping file.

Single aspects are identified by the aspect type and except Basic Property aspects, the aspect description.

### Import Mode

The import function can run in two modes, either the **Keep** mode or the **Override** mode. When instantiate an object the first time (this is when the target object doesn’t exist) keep mode and override mode have the same behavior.

Aspects are copied from the library and updated with the supplied values. This default behavior differs when re-import (this is instantiate an object, which already exists in the target project), depending on the selected import mode.

**Keep** mode

a. Aspects of the library object, which the target object doesn’t have, are copied.

b. The following aspects of the target object are updated (as far as they exist.):

Description attribute of Basic Property/Customer ID aspects (only if no empty in the data file)

Parameter aspect attributes, which fit to a Parameter aspect

Target location in the structure is updated (only if defined in the he mapping file - (see Section A.1, Overview - Instantiate Behavior of Structure Aspects).

**Override** mode

If library and target object have the same aspect (identified by type and description), the aspect of the target object is deleted and then copied from the library object again. But this is done only if the user has enabled it in the Override Options for the aspect type. If not Override mode works in the same way as Keep mode.

## 2.3 Instantiate Aspects

Instantiate aspects is copying single aspects from a library. This function is intended to instantiate more than one aspect of the same type to the same Advant object.
The data are stored in an import data file, a comma separated file (*.csv file), where each line describes one aspect to be imported/instantiated. For the complete file format see Chapter 4, Reference - Import File Formats

**NOTE**

This is not applicable for structure and basic property aspects.

### Identification of Aspects

The identification of an aspect within a project is composed of two parts.
First the Advant object, the aspect is related to has to be defined. This is done by using the object’s name (in a Basic Property aspect).
The aspect itself is identified by the aspect type (and the responsible aspect system) and either the aspect description or the aspect relative reference designation.
This means that aspect descriptions or aspect relative reference designations have to be unique within Advant objects when using this function.

### Keep and Override Mode

When running in **Keep** mode the aspect is copied from the library only if the target aspect is not available. This behavior allows to run import with the same data file several times without adding multiple aspects.
When running in **Override** mode the aspect always is deleted before instantiating.

## 2.4 Create Objects and Aspects

Create objects and aspects is creating new objects or adding aspects without using a library project. All data needed to do this, are in the import data file.

This import behavior allows creating of:

- Advant objects
- Basic Property aspects
- Structure aspects (places an object in a structure)
- Parameter aspects
- Document aspects
- Node Control Software aspects
- Circuit aspects
- Type Circuit aspects
- DBSection aspects
- DBElement aspects
- El. Circuit Diagram aspects
- El. Connection Diagram aspects
- El. Layout Diagram aspects
- El. Overview Diagram aspects.

Depending on the aspect type, you want to create different data must be provided in the import data file.
The import data file is a comma separated file (*.csv file), where one row describes the data for one object or aspect.
For details see Section 4.9, Create Aspects and Objects and Section 4.10, Create Electrical Diagram Aspects

When running the Import function in **Keep mode** first it is checked if a defined object or aspect exists. This is done for objects by searching for the defined object name. If the object is found no create takes place.
It is performed for aspects by searching for the aspect type and the aspect description. If the aspect is found no create takes place.
This behavior allows to run import with the same data file several times without creating multiple objects with the same name or to add multiple aspects to an object.

When running the Import Function in **Override mode** already existing objects and aspects are deleted. The creating takes place anyway. Deleting a structure aspects deletes all substructure aspects of it, too. This behavior allows to replace existing objects and structures with modified data.

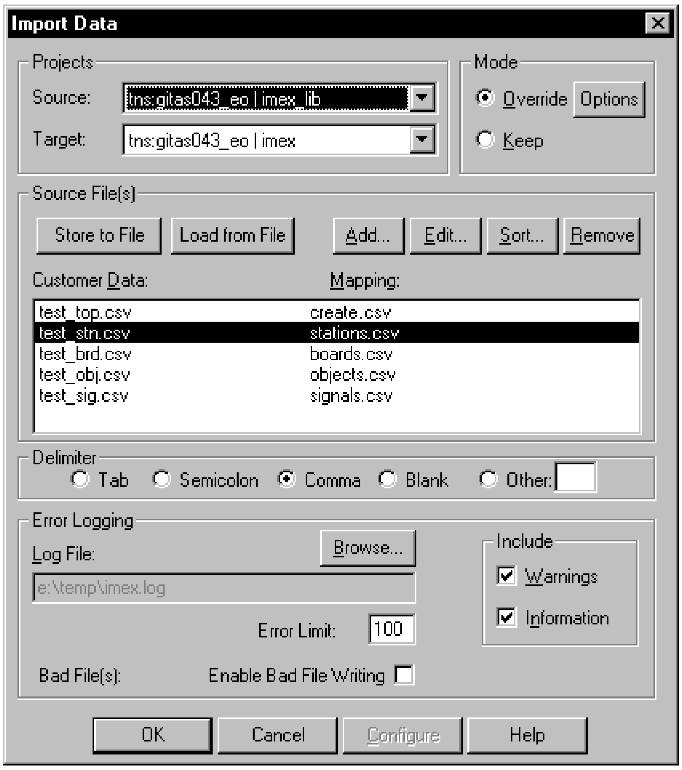
## 2.5 Create Configuration Data

For import behavior of **create configuration data** see the descriptions in Chapter 4, Reference - Import File Formats.

- Object types, Section 4.11, Create Object Types
- Structures, Section 4.12, Create Structures
- Structure display settings, Section 4.13, Create Structure Display Settings
- Document templates, Section 4.14, Create Document Templates
- Parameter types, Section 4.15, Create Parameter Aspect Types.

## 2.6 Starting the Import Function

The import function is accessed via the Structure Builder menu items **Project | Import** (see user user’s guide “AdvaBuild Structure Builder”). The menu item is visible if at least one project is opened.
The import function starts with the **Import Data** window (see Figure 2-3 below), where the import data can be entered/selected.



Source: tns:gitas043_eo | imex_lib
Target: tns:gitas043_eo | imex

● Override
○ Keep

| Customer Data: | Mapping: |
|---|---|
| test_top.csv | create.csv |
| test_stn.csv | stations.csv |
| test_brd.csv | boards.csv |
| test_obj.csv | objects.csv |
| test_sig.csv | signals.csv |

○ Tab ○ Semicolon ● Comma ○ Blank ○ Other:

Log File: e:\temp\imex.log
Error Limit: 100
☑ Warnings
☑ Information
Bad File(s): Enable Bad File Writing ☐

*Figure 2-3. Import Data Window*

### Projects

Instantiate data requires a source project and a target project. With the combo boxes for Source and Target you can select projects, opened before with the Structure Builder.

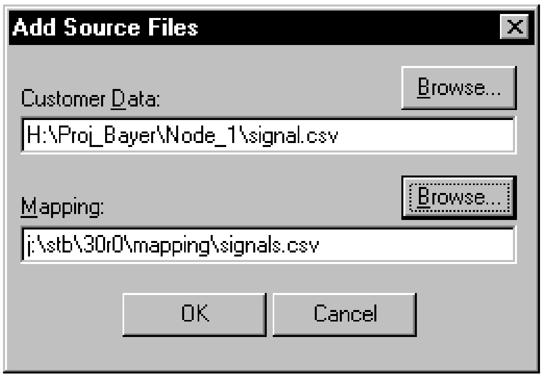
### Source File(s)

For each data file (Customer Data) which has to be imported a mapping file (Mapping) has to be defined, too. It describes the kind of data and which rules has to be executed during the import. Defined mapping files are:

| | |
|---|---|
| • **stations.csv** | Instantiate station objects. |
| • **objects.csv** | Instantiate process objects. |
| • **boards.csv** | Instantiate board/module objects. |
| • **signals.csv** | Instantiate signal objects. |
| • **caee_brd.csv**<br>**caee_brd_ard.csv** | Instantiate CAEE board objects. |
| • **caee_obj.csv**<br>**caee_obj_ard.csv** | Instantiate CAEE process objects. |
| • **caee_sig.csv**<br>**caee_sig_ard.csv** | Instantiate CAEE signal objects. |
| • **create.csv**<br>**create_ard.csv** | Create objects and aspects. |
| • **aspects.csv** | Instantiate aspects. |
| • **Create_ELDia_Aspects.csv** | Create electrical diagram aspects. |
| • **Config.csv** | Create configuration data. |
| • **RenameObjects.csv** | Rename objects |
| **RenameAspects.csv** | Rename aspects. |

The files are stored in \abtool1**stb**\release-id**mapping***.csv.
Which Mapping files has to be used is described in Chapter 4, Reference - Import File Formats.

All files in the **Source File(s)** list box are processed in the order as they appear. For adding files into the list box use the **Add** button. It launches the **Add Source Files** window (see Figure 2-4) where a single combination of data file plus mapping file can be entered manually or selected by using the **Browse** button.



Customer Data: H:\Proj_Bayer\Node_1\signal.csv

Mapping: j:\stb\30r0\mapping\signals.csv

*Figure 2-4. Add Source Files Window*

For modifying a single entry in the **Source File(s)** list box select an entry and click the **Edit** button. It launches the **Add Source Files** window, displaying the current data.

Rearranging the order in the **Source File(s)** list box can be done by using the **Sort** button. It is visible if there are at least two entries in the list.

For removing an entry from the **Source File(s)** list box select an entry and click the **Remove** button.

**Store to File** button allows to save the current setting into a Set-file. This is rejected if the Source File entries do not fulfill the Set-file requirements.

Push the **Load from File** button to load setting from an existing Set-file. See also Section 3.6, Set-Files

### Error Logging

The import function writes all messages (errors, warnings and informations) into a file (Log file). A Log file has to be defined before the import can be started. Use the Browse button to select or enter the file name.

If the defined Log file doesn’t exists, it will be created. If a file with the defined name already exists, it will be overridden. A default file extension log is added automatically, if no extension is specified.

What the import function writes to the Log file can partly controlled with the Include check boxes, where the output of Warnings and Informations can be suppressed. Errors will be printed always to the Log file. Error Limit defines the number of errors, accepted during the import over all files in the Source File(s) list. If the number of errors exceeds this limit import terminates.

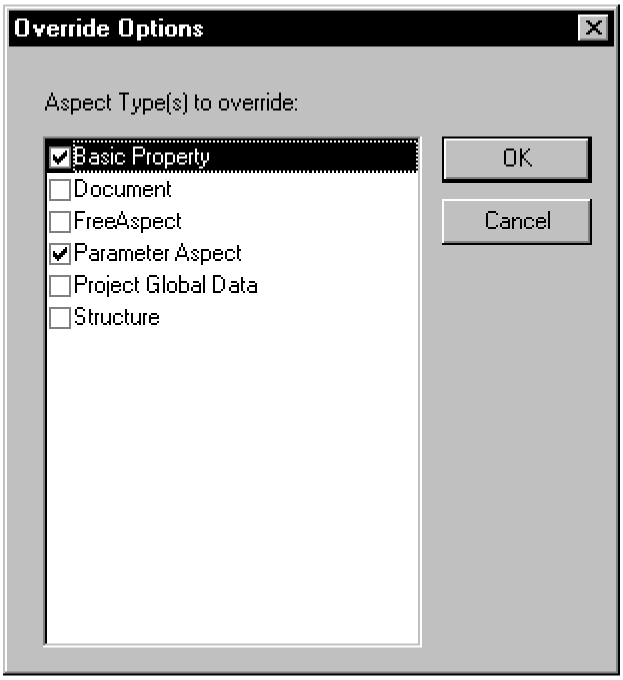
Valid range for Error Limit is 0 - 9999.

### Bad File(s)

Bad file writing can be switched on/off with the **Enable Bad File Writing** check box. Details about Bad files are described in Section 2.8, Error Logging.

### Mode

The selected Mode controls the import behavior. Select either Keep and Override for an Import run. Default is Keep mode. Both modes differs slightly depending on what kind of import is running See Section 2.2, Instantiate Objects... Section 2.4, Create Objects and Aspects for details.
Only if Override is selected, the Options button becomes active. Click the Options button to launch the Override Options window (see Figure 2-5) where you can enable/disable overriding for aspect types.



Override Options

Aspect Type(s) to override:

☑ Basic Property
☐ Document
☐ FreeAspect
☑ Parameter Aspect
☐ Project Global Data
☐ Structure

*Figure 2-5. Override Options Window*

### Delimiter

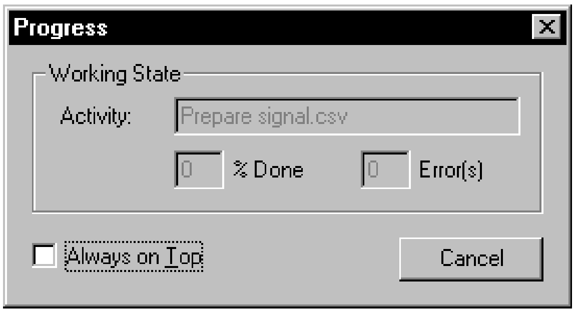
The separator token in Import Data Files (csv files) is user defined, but not all characters of the ACSII character set can be used as delimiter. Such as not printable characters cannot be selected/entered.

### Starting Import

If all required data in the Import Data window are edited/selected, the OK button becomes active. Click the OK button to start the import.

## 2.7 Import Progress

During the run time of the import function a **Progress** window (see Figure 2-6) is visible on the desktop. It displays the **Working State** of the currently running import.



Activity: Prepare signal.csv

0 % Done

0 Error(s)

*Figure 2-6. Progress Window*

### Activity

Displays which import step the import function performs. For each Source File following steps are done:

```
Prepare <Customer Data File>
Import <Customer Data File>
```

### Error(s)

Displays the number of currently raised errors.

### % Done

Displays the amount of processed data in %. This data needn’t to be identical to the elapsed time

The Progress window disappears from the desktop if the import function terminates. It is followed by the Import Exit window giving a summary message about how the import has terminated (with/without errors, canceled by user or stopped).
Then another window rises, asking for launching the Log file.

## 2.8 Error Logging

### Log file - Overview

A Log file is written each time the import function is activated. The Log file is basically organized in several sections as shown in Figure 2-7 below. The main section is the overall import section and sections for each file (file sections), which has to be imported are embedded into the main section. .

```
Main section start
     1.File section start
     1.File section end

     2.File section start
     2.File section end

     3.File section start
     3.File section end

     n.File section start
     n.File section end
Main section end
```

*Figure 2-7. Log File Sections - Principle*

### Main Section

The main section begins with the START IMPORT line, followed by the control data, which have been selected/inserted in the Import Data window (projects, mode and error logging) and which are valid for all files to import.

```
START IMPORT

     Source project.......: <library project name> (<data base>)
     Target project.......: < target project name> (<data base>)
     Import mode.........: <KEEP> or <OVERRIDE>
     Override Options....: <aspect types to override>
     Log file............: <log-file name>
     Error limit.........: <no>
     Include Information.: <YES> or <NO>
     Include Warnings....: <YES> or <NO>
     Bad-File writing....:<YES> or <NO>
     Field delimiter.....:<delimiter token>
```

The main section ends with a summary message and an END IMPORT line.

```
     <summary message>
END IMPORT
```

Possible summary messages are:

```
     terminated - no errors, no warnings
     terminated with <n> error(s), <n> warning(s)
     cancelled by user - total number of error(s) <n>
```

```
stopped - too many error(s)
stopped - log file access error
stopped - sever error
```

### File Section

For each import file a separate file section is written beginning with the file name and the selected mapping file. Followed by a date/time information and ending with an End file line.
If a bad-file is created the full file name of it is written before.

```
Import file..........: <import file name>
Mapping file.........: <mapping file name>
  Start <time and date>

  End <time and Date>     Elapsed Time: <time>

     Bad-file is created - <bad file name>
End     file.........: <import file name>
```

For each line in the import data file, an info line is put into the file section. It contains the information about what action is performed. It is composed of the line number of the import file, the target object name and the library object name.

```
Info:Line <no> Instantiate <target object> from <library object>
```

This line is followed by further information lines showing what had happened in detail. This is usually when aspects are duplicated or updated. Also error(s), which rise during computing an import data line can follow.

### Example Log File

The example below shows a Log file, created during the import of one import data file containing station data (two station objects).

```
START IMPORT

Source project.......: imex_lib (tns:gitas043_eo)
Target project.......: imex (tns:gitas043_eo)
Import mode..........: KEEP
Log file.............: e:\temp\imex.log
Error limit..........: 100
Include informations.: YES
Include warnings.....: YES
Bad-File writing.....: YES
Field delimiter......: ,

Import file..........: e:\temp\stations.csv
```

```
Mapping file.........: j:\stb\32r0a2r0\mapping\stations.csv

  Start09:32:33   13.08.1999

  Info :Line 1 Instantiate Stn_99T from AC70-X
  Error:[66] in line 1
        STBASI 207 Object AC70-X not found in Project imex_lib

  Info :Line 2 Instantiate Stn_00T from AC70
  Info :Duplicate Aspect
  Info :Duplicate Aspect
  Info :Duplicate Aspect
  Info :Duplicate Aspect
  Info :Duplicate Aspect
  Info :Update Aspect

  End09:32:38 13.08.1999Elapsed Time: 5 second(s). (0h 0min 5s)

  Bad-file is created - E:\TEMP\Bad285.tmp
End     file.........: e:\temp\stations.csv

terminated with 1 error(s), 0 warning(s)
END  IMPORT
```

### Bad-File Writing

Data lines in an import file, which can not be successfully imported are written into a bad-file. The bad-file has the same format as the import file and can be imported again. It contains an extract of the original import file.

A bad-file is not written in any case. Two conditions have to be fulfilled before the import function creates a bad-file.

1. The import function must have terminated with errors. It is shown in the summary message.

   ```
   terminated with <n> error(s), <n> warning(s)
   ```

2. No file access errors have been raised for the import file or the mapping file.

   If the import function stops, due to errors or is terminated by the user no bad-file is written. This is when the import end with one of the summary messages listed below.

   ```
   cancelled by user - total number of error(s) <n>
   stopped - too many error(s)
   stopped - log file access error
   stopped - sever error
   ```

# Chapter 3 Export

## 3.1 Export Process

The Export function extracts data from AEW projects into data storage outside AEW.
The data scope is Advant Objects and their related aspects.
The export range can be from one single object up to all objects of one project.
Exported data follow the format of the Import function and can (re-)imported into any AEW project.

The Export function is a basic mechanism of the AEW.
Extracting only data has no function until this data can be reused for any purpose.
Together with the Import function it opens possibilities to improve working with AEW.
For engineering work with AEW the Export function has two scopes.

1. Export all objects of a project.
   Object and related aspects can be extracted, and consistent re-imported into other AEW projects (reuse of solutions).
2. Export selected objects from a project.
   Object, selected in a structure of the Structure Builder, can be extracted with all their related aspects. Consistency at a re-import is not given in all cases. It depends on the relationships of the extracted data (i.e. selected objects of a node can be consistent).

   Typically inconsistency is given, if structure aspects are exported and the related parent objects are not exported.

The Export function always can be used to pass configuration data from one project to another. I.e. structure definitions, object types or project options.

Following AEW data can exported:

- Advant Objects
- Aspects:
  - Structure aspects
  - Basic Property aspects
  - Parameter aspects
  - Document aspects
  - Free aspects
  - Node Control aspects
  - Circuit aspects
  - Type Circuit aspects
  - DBSection aspects
  - DBElement aspects.

- Configuration data:
  - Object types
  - Structures
  - Structure display settings
  - Document templates
  - Parameter types.

AEW data, which cannot be exported or where export is not supported:

Project Global Data aspects.

## 3.2  Starting the Export Function

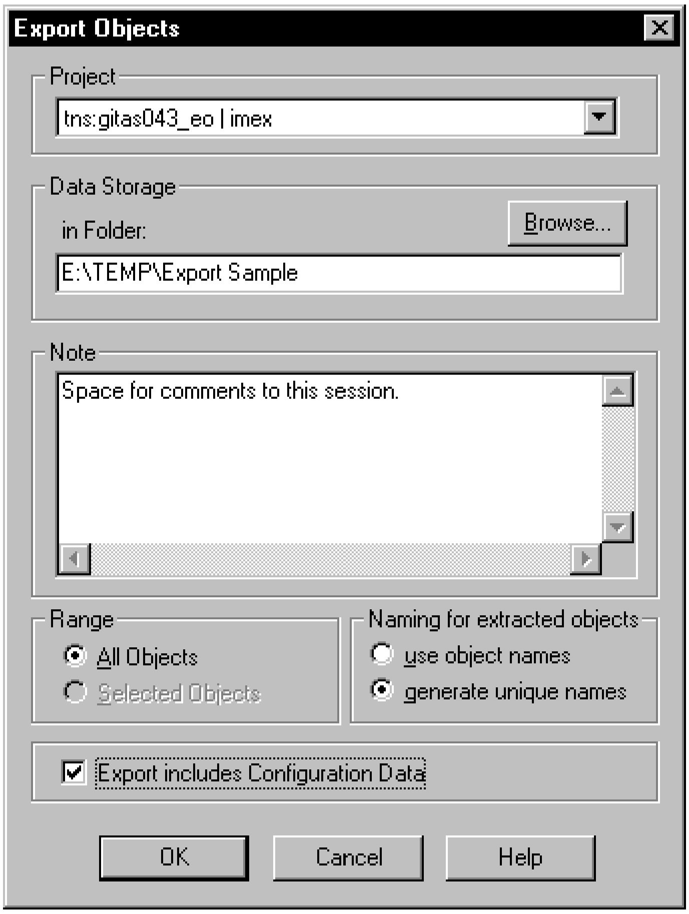
The Export function is started using the Structure Builder's menu item

**Project | Export**

Before that, objects within one structure can be selected, to define the Export range.

Alternatively nothing is selected, which means all objects of a project are in the scope.

The Export function starts with its user interface:



Project: tns:gitas043_eo | imex

Data Storage
in Folder: E:\TEMP\Export Sample

Note: Space for comments to this session.

Range
● All Objects
○ Selected Objects

Naming for extracted objects
○ use object names
● generate unique names

☑ Export includes Configuration Data

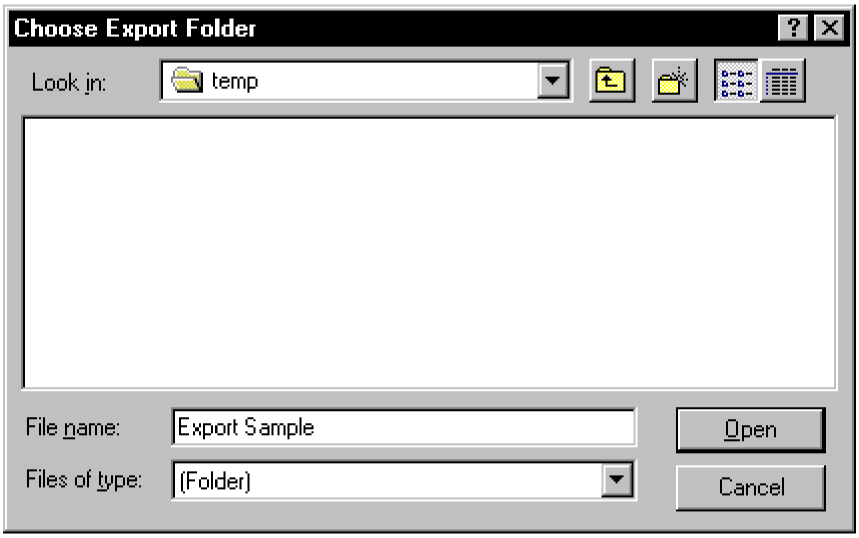
*Figure 3-1. Export Window*

### Project

All projects, which are opened in the Structure Builder are available in this project select control. If an object selection is done before, the right project is automatically selected at start-up.

### Data Storage

The export result are multiple files organized in a folder structure, which is managed by the Export function. The user has to define only the root folder, a new, empty folder.
A file selector window is launched for browsing the directories, when the **Browse** button is clicked.



File name: Export Sample

Files of type: (Folder)

*Figure 3-2. Folder Select Window*

**NOTE**

Basically the file selector window is intended to select files, not to select folders. Folders in this window are used only for navigation. **So always a new folder name has to be entered**.

### Note

Is a place for adding some notes to the export session. These note is put into the Export Log file.

### Range

Defines the range of Advant objects to be exported, dependent on how the Export has been started (with or without objects selected in the Structure Builder), the Range selection is set. It can be changed by the user.

### Naming for Extracted Objects

The user can select how object names are stored in the extracted data files.
This is very important later when (re-)importing the exported data.
To avoid conflicts with duplicate names click generate unique names
This forces the Export function to generate unique names for objects (see Section 3.7, Working with Unique Names). So exported data can be proper (re-)imported into any other project.
If the uniqueness of objects is already given, click use object names.

### Export Includes Configuration Data

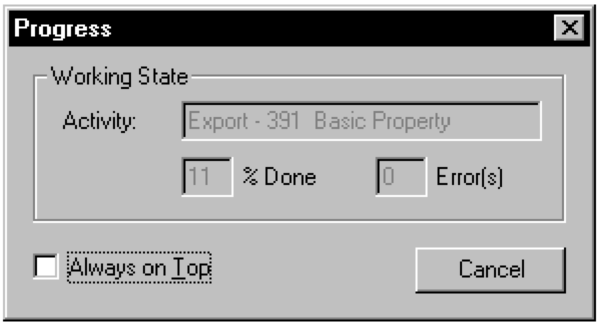
Configuration data includes:

- Object types
- Structures
- Structure display settings
- Document templates
- Parameter types.

If all required information is added/selected in the Export user interface, the OK button is enabled and the export of objects/aspects is started when clicked.

## 3.3 Processing

After it is started, the Export user interface is closed and a working box is launched, displaying the progress status of the running export function.



Progress

Working State

Activity: Export - 391 Basic Property

11 % Done

0 Error(s)

*Figure 3-3. Export Progress Window*

The Export Progress window is the same as at the Import function. The displayed activities are listed below.

### Activity

First all objects are exported.

`Export - <number> Advant Objects` … is displayed.

Afterwards all single aspect types are exported in the order as the aspect systems has been installed on the server computer.

`Export - <number> <Aspect Type>` … is displayed.

This includes also that aspect type related configuration data (if selected) are part of the export.

### Error(s)

Displays the number of raised errors.
A single error means that one of the activities has failed.

### % Done

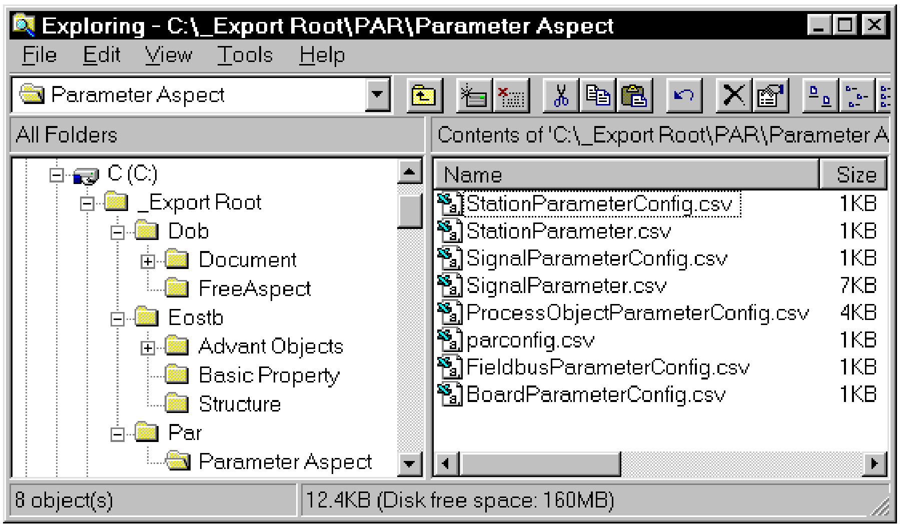
Displays the number of already performed activities in percent. It is not identical with the elapsed time.

### Cancel

The button is for cancel a running Export function. In the worst case this is possible only between two activities.

## 3.4 Output Data

Figure 3-4 shows the principle export result. The complete file information in the single folder structures depends finally on the the exported data.



| Name | Size |
|---|---|
| StationParameterConfig.csv | 1KB |
| StationParameter.csv | 1KB |
| SignalParameterConfig.csv | 1KB |
| SignalParameter.csv | 7KB |
| ProcessObjectParameterConfig.csv | 4KB |
| parconfig.csv | 1KB |
| FieldbusParameterConfig.csv | 1KB |
| BoardParameterConfig.csv | 1KB |

8 object(s)
12.4KB (Disk free space: 160MB)

*Figure 3-4. Export Output Data (Sample)*

Basically one folder is created for each aspect system, named like the aspect systems abbreviation.
For each aspect type one folder will be created below, usually named like the aspect type.
An overview of all data files, which can be created during an export process, is shown is Table 3-1. These files can be re-imported. Additionally files (i.e. document files of document aspects) or folders are not described in detail.

Important for a later use of exported data is a file named `Export.set`, located in the export folder.It contains the whole information about all exported data. In the case of re-import, the import function, which is improved for reading **Set files**, can open and read this file.

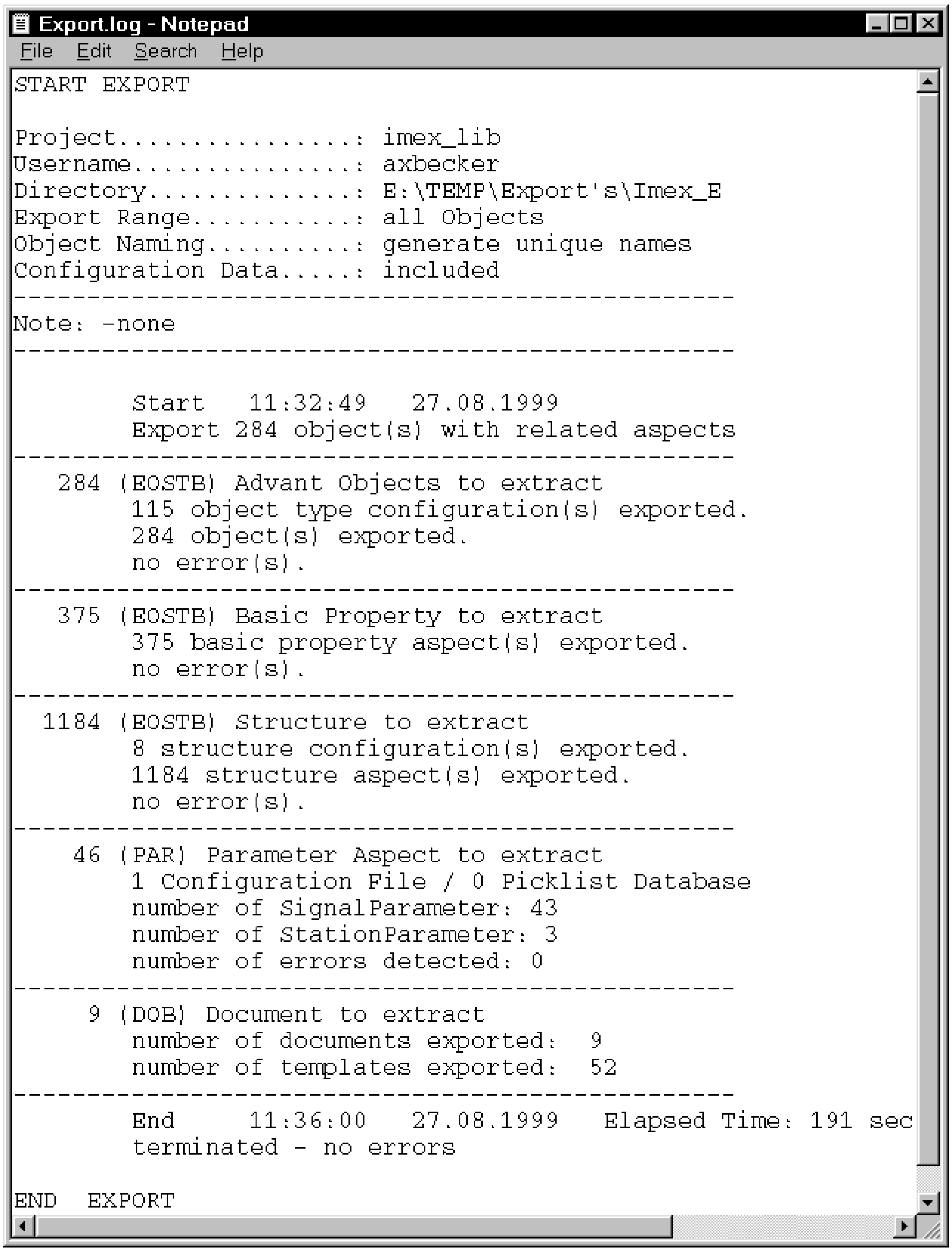
| Folder Name | File Name | Import File Format |
|---|---|---|
| EOSTB/Advant Objects | AdvantObjects.csv | Create Objects |
| | AdvantObjectsConfig.csv | Create Object Types |
| | UniqueAspects.csv | Rename Aspects |
| | UniqueNames.csv | Rename Objects |
| EOSTB/Basic Property | Basic Property.csv | Create Basic Property Aspects |
| EOSTB/Structure | Structure.csv | Create Structure Aspects |
| | StructureConfig.csv | Create Structures |
| | StructureDisplayConfig.csv | Create Structure Display Settings |
| DOD/Document | documents.csv | Create Document Aspects |
| | dobconfig.csv | Create Document Templates |
| DOB/FreeAspect | documents.csv | Create Free Aspects |
| PAR/Parameter Aspect | <Parameter>.csv * | Create Parameter Aspect |
| | <Parameter>Config.csv * | Create Parameter Aspect Type |
| EOAPB/Node Control Software | Node Control Software.csv | Create Node Control Software Aspects |
| EOAPB/Circuit | Circuit.csv | Create Circuit Aspects |
| EOAPB/Type Circuit | Type Circuit.csv | Create Type Circuit Aspects |
| EOAPB/DBSection | DBSection.csv | Create DBSection Aspects |
| EOAPB/DBElement | DBElement.csv | Create DBElement Aspects |

*Table 3-1. Overview Export Files*

* File format depends on the parameter aspect type. For every parameter aspect separate files are created, named like the parameter aspect type.

## 3.5 Export Logging

The export process creates a Log file, containing a summary about the export process. It is named **`Export.log`** and located in the export folder. Figure 3-5 shows an sample.



```
START EXPORT

Project................: imex_lib
Username...............: axbecker
Directory..............: E:\TEMP\Export's\Imex_E
Export Range...........: all Objects
Object Naming..........: generate unique names
Configuration Data.....: included
-------------------------------------------------
Note: -none
-------------------------------------------------


        Start   11:32:49   27.08.1999
        Export 284 object(s) with related aspects
-------------------------------------------------
   284 (EOSTB) Advant Objects to extract
        115 object type configuration(s) exported.
        284 object(s) exported.
        no error(s).
-------------------------------------------------
   375 (EOSTB) Basic Property to extract
        375 basic property aspect(s) exported.
        no error(s).
-------------------------------------------------
  1184 (EOSTB) Structure to extract
        8 structure configuration(s) exported.
        1184 structure aspect(s) exported.
        no error(s).
-------------------------------------------------
    46 (PAR) Parameter Aspect to extract
        1 Configuration File / 0 Picklist Database
        number of SignalParameter: 43
        number of StationParameter: 3
        number of errors detected: 0
-------------------------------------------------
     9 (DOB) Document to extract
        number of documents exported:  9
        number of templates exported:  52
-------------------------------------------------
        End     11:36:00   27.08.1999   Elapsed Time: 191 sec
        terminated - no errors

END  EXPORT
```

*Figure 3-5. Export Log file (Sample)*

## 3.6 Set-Files

A Set file contains the settings, entered/selected in the **Import Data window** (see Figure 2-3).

Set files can be created in the Import Data window with the **Store to File** button and can be reloaded with the **Load from File** button.

Set-Files are created from the **Export function**, too. So re-import of once exported data is easy to do. Just reload the Set file and start the Import function. The Set file contains all import Data files and the related mapping files in the right order.
Set files, created by the Export Function, are named **`Export.set`** and located in the Export Folder.

### Limitation

Data file entries in a Set file have to be located

1. in a hierarchical relationship
2. on the same level or below the actual Set-File.

## 3.7 Working with Unique Names

When running the Import function to create objects and aspects, object names and aspect description are used for identification. Whenever the uniqueness of this data is not given, this can lead to unexpected import results.

To ensure that all objects and their aspects, which should be exported, could be completely re-imported in any other project, export has to be performed with the **generate unique names** option.

In this case all object names and aspect descriptions in the data files are substituted with a unique identifiers during export. Additionally reference lists are created, where the relation between object names / aspect descriptions and the unique identifiers is stored.
These lists are the files `UniqueNames.csv`, `UniqueAspects.csv`, placed in the export folder, which can be treated as normal import data files.

When re-import data, created with the Export function, you usually load the Set file in the Export folder. If the Export was done with the **generate unique names** option the two files `UniqueNames.csv` and `UniqueAspects.csv` are part of the Set file, placed at the end of the list (see Figure 3-6).

After Import function has computed all files in the set, with the last two entries:

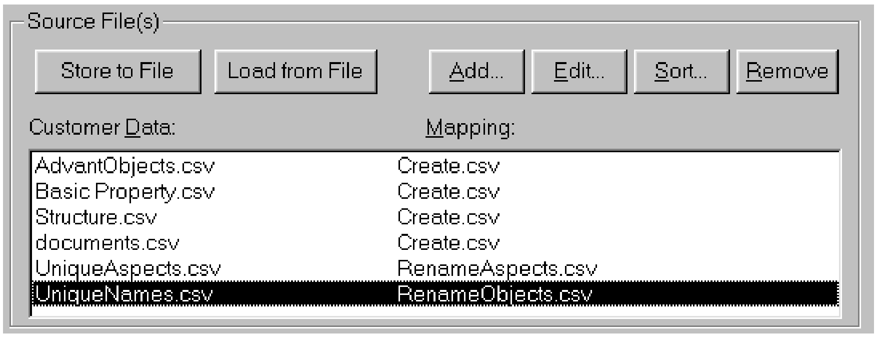
```
UniqueAspects.csv   -     RenameAspects.csv
UniqueNames.csv     -     RenameObjects.csv
```

the created objects and aspects are renamed with their originally names.

Import stops, if at least one error rises in the former processing.

**NOTE**

All data files have to be imported without any error, before the renaming should be done.



| Customer Data: | Mapping: |
|---|---|
| AdvantObjects.csv | Create.csv |
| Basic Property.csv | Create.csv |
| Structure.csv | Create.csv |
| documents.csv | Create.csv |
| UniqueAspects.csv | RenameAspects.csv |
| UniqueNames.csv | RenameObjects.csv |

*Figure 3-6. Import - Rename Files*

# Chapter 4 Reference - Import File Formats

## 4.1 Instantiate Signal Objects

To instantiate signal objects from a library project.

Use this import file format with mapping **signals.csv**.

*Table 4-1. File Format of Instantiate Signal Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1 | 1.st Library reference. |
| FuncParent | For this reference placing the object in the Function Structure is enabled. The parent object identification is defined in attribute FuncParent. |
| Node<br>Designation<br>Channel | Placing the object in OCS structure is enabled, too. The parent object identification is defined in attributes Node, Designation and Channel. |
| Libref2 | 2nd Library reference. |
| <par1...par n> | Signal Parameter attributes |

## 4.2 Instantiate Station/Fieldbus Objects

To instantiate station objects from a library project.

Use this import file format with mapping **stations.csv**.

*Table 4-2. File Format of Instantiate Station Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1 | 1.st Library reference. |
| OCSParent | For this reference placing the object in the OCS structure is enabled. The parent object identification is defined in attribute OCSParent. |

*Table 4-2. File Format of Instantiate Station Objects*

| Attribute | Interpretation |
|---|---|
| Libref2 | 2nd Library reference. |
| Designation | Relative Reference Designation in the OCS Structure |
| <par...par n> | Station/Fieldbus Parameter attributes |

## 4.3 Instantiate Process Objects

To instantiate process objects from a library project.

Use this import file format with mapping **objects.csv**.

*Table 4-3. File Format of Instantiate Process Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1<br>FuncParent<br>OCSParent | 1.st Library reference.<br>For this reference placing the object in the Function Structure is enabled. The parent object identification is defined in attribute FuncParent.<br>Substructure objects are created.<br>For this reference placing the object in the OCS structure is enabled. The parent object identification is defined in attribute OCSParent. |
| Libref2 | 2nd Library reference. |
| OCSDesig | Relative Reference Designation in the OCS Structure. |
| <par1...par n> | Process Object Parameter attributes |

## 4.4 Instantiate Board/Module Objects

To instantiate board/module objects from a library project.

Use this import file format with mapping **boards.csv**.

*Table 4-4. File Format of Instantiate Board/Module Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1<br>OCSParent | 1.st Library reference.<br>For this reference placing the object in the OCS Structure is enabled. The parent object identification is defined in attribute OCSParent. Substructure objects are created. |
| Designation | Relative Reference Designation in the OCS Structure |
| Libref2 | 2nd Library reference. |
| <par1...par n> | Board Parameter attributes |

## 4.5 Instantiate CAEE Signal Objects

To instantiate signal objects from a library project.

Use this import file format with mapping **caee_sig.csv** or **caee_sig_ard.csv.**

**caee_sig_ard.csv** requires absolute reference designations instead of object names in attributes FuncParent and Node. The absolute reference designation entered in the file has to fit to the structure settings on your client PC.

*Table 4-5. File Format of Instantiate CAEE Signal Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1<br>FuncParent<br>Node<br>Designation<br>Channel | 1.st Library reference.<br>For this reference placing the object in the Function Structure is enabled. The parent object identification is defined in attribute FuncParent.<br>Placing the object in OCS structure is enabled, too. The parent object identification is defined in attributes Node, Designation and Channel. |

*Table 4-5. File Format of Instantiate CAEE Signal Objects*

| Attribute | Interpretation |
|---|---|
| FuncDesig | Relative Reference Designation in the Function Structure. |
| Libref2 | 2nd Library reference. |
| <par1...par n> | Signal Parameter attributes |

## 4.6 Instantiate CAEE Process Objects

To instantiate process objects from a library project.

Use this import file format with mapping **caee_obj.csv** or **caee_obj_ard.csv.**

**caee_obj_ard.csv** requires absolute reference designations instead of object names in attributes FuncParent, OCSParent, ProdParent and MLocParent. The absolute reference designations entered in the file has to fit to the structure settings on your client PC.

*Table 4-6. File Format of Instantiate CAEE Process Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1 | 1.st Library reference. |
| FuncParent | For this reference placing the object in the Function Structure is enabled. The parent object identification is defined in attribute FuncParent.<br>Substructure objects are created. |
| OCSParent | Placing the object in OCS structure is enabled. The parent object identification is defined in attribute OCSParent.<br>Substructure objects are created. |
| ProdParent | Placing the object in Product Structure is enabled. The parent object identification is defined in attribute ProdParent.<br>Substructure objects are created. |
| MLocParent | Placing the object in Mounting Location Structure is enabled. The parent object identification is defined in attribute MLocParent.<br>Substructure objects are created. |
| FuncDesig | Relative Reference Designation in the Functional Structure. |
| OCSDesig | Relative Reference Designation in the OCS Structure. |

*Table 4-6. File Format of Instantiate CAEE Process Objects*

| Attribute | Interpretation |
|---|---|
| ProdDesig | Relative Reference Designation in the Product Structure. |
| MLocDesig | Relative Reference Designation in the Mounting Location Structure. |
| Libref2 | 2nd Library reference. |
| <par1...par n> | Process Object Parameter attributes |

## 4.7 Instantiate CAEE Board Objects

To instantiate board objects from a library project.

Use this import file format with mapping **caee_brd.csv** or **caee_brd_ard.csv.**

**caee_brd_ard.csv** requires absolute reference designations instead of object names in attributes OCSParent, ProdParent and MLocParent. The absolute reference designations entered in the file has to fit to the structure settings on your client PC.

*Table 4-7. File Format of Instantiate CAEE Board Objects*

| Attribute | Interpretation |
|---|---|
| Name | Advant object name |
| Description | Description |
| LibRef1 | 1.st Library reference. |
| OCSParent | For this reference placing the object in the OCS Structure is enabled. The parent object identification is defined in attribute OCSParent.<br>Substructure objects are created. |
| ProdParent | Placing the object in Product Structure is enabled. The parent object identification is defined in attribute ProdParent.<br>Substructure objects are created. |
| MLocParent<br>Rack<br>Slot | Placing the object in Mounting Location Structure is enabled. The parent object identification is defined in attributes MLocParent, Rack and Slot.<br>Substructure objects are created. |
| OCSDesig | Relative Reference Designation in the OCS Structure. |
| ProdDesig | Relative Reference Designation in the Product Structure. |

*Table 4-7. File Format of Instantiate CAEE Board Objects*

| Attribute | Interpretation |
|---|---|
| Libref2 | 2nd Library reference. |
| <par1...par n> | Board Parameter attributes |

## 4.8 Instantiate Aspects

To instantiate single aspects from a library object.

Use this import file format with mapping **aspects.csv**.

*Table 4-8. File Format of Instantiate Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Name of the Advant object where the aspect has to be added. |
| EONameCategory | Name category of the Advant object. If this attribute is empty the default *Customer ID* is assumed. |
| AspType | Name of the aspect type, which has to be instantiated, as it is displayed in the Structure Builder i.e. *Parameter Aspect.* |
| AspSys | The short name of the aspect system, which manages the aspect type:<br>Aspect system AspSys<br>Structure Builder *EOSTB*<br>Parameter Builder *PAR*<br>Document Builder *DOB*<br>Application Builder *EOAPB*<br>Electrical Diagram Builder *ECAE* |
| AspDesc | Depending on the attributes LibAspDesc and LibAspRelRef the value is used to identify the target aspect.<br>At instantiation the value is set to the aspect *Description* attribute. |
| AspRelRef | Depending on the attributes LibAspDesc and LibAspRelRef the value is used to identify the target aspect.<br>At instantiation the value is set to the *Aspect Relative Reference Designation* attribute. |
| LibRef | Name of the Advant object from where the aspect has to be instantiated.<br>The basic property category has to be the same as for the target object in EOName. |

*Table 4-8. File Format of Instantiate Aspects*

| Attribute | Interpretation |
|---|---|
| LibAspDesc<br>LibAspRelRef | Only one of both attributes must have a value, the other has to be empty. Depending on which of them is filled, the identification of source and target aspect is defined.<br>LibAspDesc to identify the aspects by the Description.<br>LibAspRelRef to identify the aspects by the *Aspect Relative Reference Designation.* |
| <par1...par n> | Parameter attributes |

## 4.9 Create Aspects and Objects

To create objects or aspects of any type.

Use this import file format with mapping **Create.csv.**

*Table 4-9. File Format of Create Aspects and Objects*

| Attribute | Interpretation |
|---|---|
| EOName | These attributes has to be available in the data file.<br>How they are interpreted and what has to be filled in depends on the aspect type to create.<br>The definitions for the different aspect types are described in the following sections. |
| EONameCategory | |
| EOType | |
| AspectSystem | |
| AspectType | |
| Category | |
| AspectDescription | |
| Data1 | |
| Data2 | |
| Data3 | |
| Data4 | |
| Data5 | |
| <par1...par n> | Additional attributes can be omitted in the data file, if they are not used. |

### 4.9.1 Create Advant Objects

Two methods are provided to create Advant objects. They have a nearly identical import file format but differ in the interpretation of the **object identification**.

1. Object is identified by **Name**

   Two aspects are created. A structure aspect, which makes the object visible in the project pool structure and a basic property aspect, which keeps the object name.
   Table 4-10 shows which data has to be filled in.
   Use this import file format with mapping **Create.csv**.

*Table 4-10. Attribute Definitions - Create Advant Objects (Name)*

| **Attribute** | **Interpretation** |
|---|---|
| EOName | Name of the Advant Object. |
| EONameCategory | The category of the basic property aspect. If empty, **`Customer ID`** is assumed. |
| EOType | The object type (i.e. `Board, Signal`). |
| AspectSystem | |
| AspectType | Value has to be **`EO`**. |
| AspectDescription | Description of the basic property aspect (can be empty) |

2. Object is identified by **Absolute Reference Designation** (ARD).

   Tree aspects are created.
   A structure aspect, which makes the object visible in the project pool structure,
   a basic property aspect, which keeps the object name.
   A structure aspect in a definable structure, which keeps the identifier (ARD). The absolute reference designation entered in the file has to fit to the structure settings on your client PC.

   Table 4-11 shows which data has to be filled in.
   Use this import file format with mapping **Create_ard.csv**.

*Table 4-11. Attribute Definitions - Create Advant Objects (ARD)*

| **Attribute** | **Interpretation** |
|---|---|
| EOName | The ARD of the object (Identifier of the object). |
| EONameCategory | Structure name, where the Identifier is stored. |
| EOType | The object type (i.e. `Board, Signal`). |
| AspectSystem | |
| AspectType | Value has to be `EO.` |
| AspectDescription | Description of the basic property aspect (can be empty) |
| Data1 | ARD (identifier) of the parent object in the structure. |

*Table 4-11. Attribute Definitions - Create Advant Objects (ARD)*

| Attribute | Interpretation |
|---|---|
| Data2 | Name category of the Basic Property aspect. If empty, **`Customer ID`** is assumed. |
| Data3 | (optional) Structure Prefix |
| Data4 | Relative Reference Designation in the structure. |
| Data5 | (optional) Structure Control Flag |
| Data6 | Name of the object. - Is treated |

## 4.9.2 Create Structure Aspects

An existing Advant object is placed in a structure. Table 4-12 shows which data has to be filled in. Use this import file format with mapping **Create.csv**.

*Table 4-12. Attribute Definitions - Create Structure Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| EOType | The object type (i.e. `Board, Signal`). |
| AspectType | Value has to be **`Structure`**. |
| Category | Structure name (i.e. `= Function Structure`). |
| AspectDescription | Description for the aspect (can be empty) |
| Data1 | Name of the parent object in the target structure. If empty, no parent object is assumed and the object is placed on root level. |
| Data2 | Category of the basic property, where the parent object (defined in Data1) is in. If empty, **`Customer ID`** is assumed. |
| Data3 | Prefix value of the structure aspect (can be empty). |
| Data4 | Relative reference designation value of the structure aspect (can be empty). |
| Data5 | Control flag of the structure aspect (can be empty). |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. Use than mapping **Create_ard.csv**.

## 4.9.3 Create Basic Property Aspects

To add basic property aspects to an existing Advant object. Table 4-13 shows which data has to be filled in.

Use this import file format with mapping **Create.csv**.

*Table 4-13. Attribute Definitions - Create Basic Property Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| EOType | The object type value (i.e. `Board, Signal`). |
| AspectType | Value has to be **`Basic Property`**. |
| Category | Category of the basic property aspect. If empty, **`Customer ID`** is assumed |
| AspectDescription | Description for the aspect (can be empty) |
| Data1 | Name of the basic property aspect. If empty the default value from **EOName** is used. |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. Use than mapping **Create_ard.csv**.

## 4.9.4 Create Document Aspects

To add document aspects to an existing object. Concerning the document itself this can be either using an available document template (described in Table 4-14) or using the content of an existing document file, outside AEW (described in Table 4-15).

Use this import file format with mapping **Create.csv**.

*Table 4-14. Attribute Definitions - Create Document Aspects, Using Templates*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`DOB`**. |
| AspectType | Value has to be **`Document`**. |

*Table 4-14. Attribute Definitions - Create Document Aspects, Using Templates*

| Attribute | Interpretation |
|---|---|
| Category | Category of the basic property aspect. If empty, **`Customer ID`** is assumed |
| AspectDescription | Description for the aspect (can be empty) |
| Data1 | Document type (i.e. `Word Document`). |
| Data2 | Document suffix (i.e. `doc`). |
| Data3 | Document language (i.e. `EN`). |

*Table 4-15. Attribute Definitions - Create Document Aspects, Using External File*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`DOB.`** |
| AspectType | Value has to be **`Document.`** |
| Category | Category of the basic property aspect. If empty, **`Customer ID`** is assumed |
| AspectDescription | Description for the aspect (can be empty) |
| Data4 | Document file (i.e. `c:\temp\file.doc`). |
| Data5 | A flag identifying if the document file should be copied (value is **`COPY`**) or referenced (value is **`REF`**). If empty, COPY is assumed. |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. Use than mapping **Create_ard.csv**.

## 4.9.5 Create Parameter Aspects

To add Parameter aspects to an existing object. Table 4-16 shows which data has to be filled in.

Use this import file format with mapping **Create.csv**.

Independent of the import mode (Keep or Override), parameter attributes are always updated.

*Table 4-16. Attribute Definitions - Create Parameter Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`PAR.`** |
| AspectType | Value has to be **`Parameter Aspect.`** |
| Category | Parameter type (i.e. **`Signal Parameter`** |
| AspectDescription | Description for the aspect (can be empty) |
| <par1...par n> | The attributes of the parameter aspect are optional.<br>If attributes are defined in the data file (this means the attribute name is identical with an attribute of the related parameter aspect), its value will used at creating time. |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv**.

## 4.9.6  Create Node Control Software Aspects

To add Node Control Software aspects to an existing object. Table 4-17 shows which data has to be filled in. Use this import file format with mapping **Create.csv**.

*Table 4-17. Attribute Definitions - Create Node Control Software Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`EOAPB.`** |
| AspectType | Value has to be **`Node Control Software.`** |
| AspectDescription | Description for the aspect (can be empty) |
| NetNode | For their meaning and allowed values see *AMPL Control Configuration Commands for Structure Builder User's Guide*. |
| BusStation | |
| RNetRNode | |
| Position | |
| FCBVersion | |
| NodeType | |
| Comment | |
| Option1...Option15 | |
| NodeName | |
| PCProgItem | |
| InstanceName | |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv**.

### 4.9.7 Create Circuit Aspects

To add Circuit aspects to an existing object. Table 4-18 shows which data has to be filled in.

Use this import file format with mapping **Create.csv**.

*Table 4-18. Attribute Definitions - Create Circuit Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`EOAPB.`** |
| AspectType | Value has to be **`Circuit.`** |
| AspectDescription | Description for the aspect (can be empty) |
| Data4 | For their meaning and allowed values see *AMPL Control Configuration Commands for Structure Builder User’s Guide*. |
| FCBVersion | |
| NodeType | |
| Comment | |
| Option1...Option15 | |
| NodeName | |
| PCProgItem | |
| InstanceName | |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv**.

## 4.9.8 Create Type Circuit Aspects

To add Type Circuit aspects to an existing object. Table 4-19 shows which data has to be filled in. Use this import file format with mapping **Create.csv**.

*Table 4-19. Attribute Definitions - Create Type Circuit Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`EOAPB.`** |
| AspectType | Value has to be Type **`Circuit.`** |
| AspectDescription | Description for the aspect (can be empty) |
| Data4 | For their meaning and allowed values see *AMPL Control Configuration Commands for Structure Builder User's Guide*. |
| FCBVersion | |
| NodeType | |
| Comment | |
| Option1...Option15 | |
| NodeName | |
| PCProgItem | |
| InstanceName | |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv.**

## 4.9.9 Create DBSection Aspects

To add DBSection aspects to an existing object. Table 4-20 shows which data has to be filled in.

Use this import file format with mapping **Create.csv**.

*Table 4-20. Attribute Definitions - Create DBSection Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`EOAPB.`** |
| AspectType | Value has to be **`DBSection.`** |
| AspectDescription | Description for the aspect (can be empty) |
| Data4 | For their meaning and allowed values see *AMPL Control Configuration Commands for Structure Builder User's Guide*. |
| FCBVersion | |
| NodeType | |
| Comment | |
| Option1...Option15 | |
| NodeName | |
| PCProgItem | |
| InstanceName | |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv**.

## 4.9.10 Create DBElement Aspects

To add DBElement aspects to an existing object. Table 4-21 shows which data has to be filled in.

Use this import file format with mapping **Create.csv**.

*Table 4-21. Attribute Definitions - Create DBElement Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Used to identify the Advant object by its name (in EOName) and the category of the basic property (in EONameCategory), where the name is in. If basic property is empty, **`Customer ID`** is assumed. |
| EONameCategory | |
| AspectSystem | Value has to be **`EOAPB.`** |
| AspectType | Value has to be **`DBElement.`** |
| AspectDescription | Description for the aspect (can be empty) |
| Data4 | For their meaning and allowed values see *AMPL Control Configuration Commands for Structure Builder User’s Guide*. |
| FCBVersion | |
| NodeType | |
| Comment | |
| Option1...Option15 | |
| NodeName | |
| PCProgItem | |
| InstanceName | |

Object identification can also be done by ARD instead of the object name.
In that case EOName defines the ARD of the object and EONameCategory defines the structure of the ARD. An additional attribute Data6 is required, keeping the object name.
Use than mapping **Create_ard.csv**.

## 4.10 Create Electrical Diagram Aspects

To add Electrical Diagram aspects to an existing object. Table 4-22 shows which data has to be filled in.

Use this import file format with mapping **Create_ELDia_Aspects.csv**.

*Table 4-22. File Format of Create Electrical Diagram Aspects*

| Attribute | Interpretation |
|---|---|
| EOName | Name of Signal Advant Object |
| EONameCategory | Category of EOName, default (if empty) is Customer ID |
| AspType | Type of diagram aspect (as shown in Structure Builder)<br>(**`El. Circuit Diagram, El. Connection Diagram, El. Layout Diagram, El. Overview Diagram`**) |
| AspDesc | Aspect description to be set for new created aspect (is used also to identify aspect within later imports, if LibAspDesc is set) |
| AspRelRef | Aspect relative reference designation to be set for new created aspect |
| Optional columns within import file | |
| DOC_ID | Document Id/Number |
| SHEET_NO | Sheet Number (related to Document number) |
| SHEETNEXT | Number of next sheet |
| SHEETNOB | Sheet Number (related to function ref.des) |
| TITLE_1 | Title 1 |
| TITLE_2 | Title 2 |
| TITLE_3 | Title 3 |
| TITLE_4 | Title 4 |
| DOC_DES | Document kind designation code |
| DOC_DES1 | Document designation 1 |
| DOC_DES2 | Document designation 2 |
| DOC_TYPE | Document type |
| RESP_DEP | Responsible department |
| OWNER | Owner of document |
| BASED_ON | document based on |
| REPLACES | document replaces |

*Table 4-22. File Format of Create Electrical Diagram Aspects*

| Attribute | Interpretation |
|---|---|
| REPLACED_BY | document replaced by |
| LANGUAGE | Language code (2) |
| STANDARD | Standard (3) |
| CUST_DIS | Customer disposition |
| PREP_DATE | Prepared date |
| PREP_NAME | Prepared name |
| CHK_DATE | Checked date |
| CHK_NAME | Checked name |
| FILE_NAME | internal file name (can included subdirectories as extension to project directory path) |
| EXT_DIRECTORY | directory path of drawing to be attached |
| EXT_FILE_NAME | file name of drawing to be attached during import |
| MODE | Database connect mode for drawing<br>(Y: on-line, B: batch, N: off-line) |

## 4.11 Create Object Types

Import data of this format does the same as you can do with Structure Builder's menu item **Options | Customize | Object Types** button **New**. It creates a new object type.

Use this import file format with mapping **Config.csv**.

*Table 4-23. File Format of Create Object Types*

| Attribute | Interpretation |
|---|---|
| AspectSystem | The short name of the aspect system, which manages object types. This is the Structure Builder and the value has to be **`EOSTB`** |
| AspectType | The concerned aspect type is **`Basic Property`** |
| Category | empty - not used |
| ObjectType | Name of the object type. |
| ObjectIcon | File name of the object types icon. Only the file name has to be put in here. The file location has to be in a directory (named **`\bmp\`**) below the actual import file. |

**Example:**

*Table 4-24. Example - File Format of Create Object Types*

| AspectSystem | AspectType | Category | ObjectType | ObjectIcon |
|---|---|---|---|---|
| EOSTB | Basic Property | | Generator | generato.bmp |

## 4.12 Create Structures

Import data of this format does the same as you can do with Structure Builder’s menu item **Window | Structure | Open Structure** button **Add**. It creates a Structure.

A Structure is created only if it doesn’t exit. There is not difference between using the **Keep** or **Override** mode.

Use this import file format with mapping **Config.csv**.

*Table 4-25. File Format of Create Structures*

| Attribute | Interpretation |
|---|---|
| AspectSystem | The short name of the aspect system, which manages object types. This is the Structure Builder and the value has to be **`EOSTB`** |
| AspectType | The concerned aspect type is **`Structure`** |
| Category | empty - not used |
| StructureWindow | Name of the structure. |

**Example:**

*Table 4-26. Example - File Format of Create Structures*

| AspectSystem | AspectType | Category | ObjectType |
|---|---|---|---|
| EOSTB | Structure | | == OCS-Structure |

## 4.13 Create Structure Display Settings

Import data of this format does the same as you can do with Structure Builder's menu item **Options | Project | Set Options** button**s Add**. It creates new Structure Display Settings.

A Structure Display Setting is created only if it doesn't exit. There is not difference between using the **Keep** or **Override** mode.

Use this import file format with mapping **Config.csv**

*Table 4-27. File Format of Create Structure Display Settings*

| Attribute | Interpretation |
|---|---|
| AspectSystem | The short name of the aspect system, which manages structures. This is the Structure Builder and the value has to be **`EOSTB`** |
| AspectType | The concerned aspect type is **`Structure`** |
| Category | empty - not used |
| StructureDisplayFile | Name of a file containing the Settings. Only the file name has to be put in here. The file location has to be in the same directory as the actual import file. |

**Example:**

*Table 4-28. Example - File Format of Create Structure Display Settings*

| AspectSystem | AspectType | Category | StructureDisplayFile |
|---|---|---|---|
| EOSTB | Structure |  | StbExtract.ini |

## 4.14 Create Document Templates

Import data of this format does the same as you can do with Document Builder's menu item **Document | Options | Import Templates**. It creates Document templates.

A Document Template is created only if it doesn't exit. There is not difference between using the **Keep** or **Override** mode.

Use this import file format with mapping **Config.csv**

*Table 4-29. File Format of Create Document Templates*

| Attribute | Interpretation |
|---|---|
| AspectSystem | The short name of the aspect system, which manages documents. This is the Document Builder and the value has to be **`DOB`** |
| AspectType | The concerned aspect type is **`Document`** |
| Category | empty - not used |
| TemplatesFile | Name of a file containing the Template definitions. Only the file name has to be put in here. The file location has to be in the same directory as the actual import file. |

**Example:**

*Table 4-30. Example - File Format of Create Document Templates*

| AspectSystem | AspectType | Category | TemplatesFile |
|---|---|---|---|
| DOB | Document | | templates.csv |

## 4.15 Create Parameter Aspect Types

Import data of this format does the same as you can do with Parameter Builder's menu item **Edit | Aspect | Type Configuration** button **Add**. It creates Parameter aspect types.

An Parameter aspect type is created only if it doesn't exit.
Single attributes are added if they do not exit. Existing attribute types cannot be changed.

There is not difference between using the **Keep** or **Override** mode.

Use this import file format with mapping **Config.csv**

*Table 4-31. File Format of Create Parameter Types*

| Attribute | Interpretation |
|---|---|
| AspectSystem | The short name of the aspect system, which manages Parameter. This is the Parameter Builder and the value has to be **`PAR`** |
| AspectType | The concerned aspect type is **`Parameter Aspect`** |
| Category | empty - not used |
| TemplatesFile | Name of a file containing the Parameter definitions. Only the file name has to be put in here. The file location has to be in the same directory as the actual import file. |

**Example:**

*Table 4-32. Example - File Format of Create Parameter Types*

| AspectSystem | AspectType | Category | TemplatesFile |
|---|---|---|---|
| PAR | Parameter Aspect | | parameter.csv |

## 4.16 Create Document Objects

This format is customized to create Advant objects, which are placed in a structure and have a document aspect.
The format is a combination of the formats **Create Advant Objects** and **Create Document Aspects**.

The object is identified by ARD.

File of this format has to imported twice. First with mapping **Create_ard.csv** and than with mapping **Create_Doc_ard.csv**

*Table 4-33. File Format of Create Document Objects*

| Attribute | Interpretation |
|---|---|
| EOName | ARD of the object. |
| EONameCategory | Structure where ARD is defined. |
| EOType | Object type. |
| AspectSystem | Value is **`EOSTB`**. |
| AspectType | Value is **`EO`**. |
| Category | not used. |
| AspectDescription | Description for the aspect (can be empty) |
| Data1 | ARD of the parent object.<br>Empty means on root level |
| Data2 | Name category for the Basic Property aspect.<br>Default is **`Customer ID`**. |
| Data3 | (optional)Structure prefix. |
| Data4 | (optional) Structure Relative Reference Designation. |
| Data5 | (optional) Structure control flag. |
| Data6 | (optional) Name of the object. |
| Data7 | Document type. (When using a document template) |
| Data8 | Document suffix. (When using a document template) |
| Data9 | Document language. (When using a document template) |
| Data10 | Document file name. (When using an external document) |
| Data11 | Documents creation flag (**`COPY`** or **`REF`**). (When using an external document) |

## 4.17 Rename Aspects

To update the **Description** value of an aspect. Files of this format are created from the **Export** function, when working with unique names and added to the Set files. Table 4-34 lists the format. This format is not intended for manual creating. This import file format is used with mapping **RenameAspects.csv**.

*Table 4-34. Attribute Definitions - Rename Aspects*

| Attribute | Interpretation |
|---|---|
| UUIDName | Used to identify the Advant object by its unique name (UUIDName) and the category of the basic property (EONameCategory), where the name is in. |
| EONameCategory | |
| AspectDescription | The new description value. |
| AspectSystem | Value is EOSTB. |
| AspectType | Aspect Type. |
| UUIDDescription | The old description value |

## 4.18 Rename Objects

To rename objects. Files of this format are created from the **Export** function, when working with unique names and added to the Set files. Table 4-35 lists the format. This format is not intended for manual creating. This import file format is used with mapping **RenameObjects.csv**.

*Table 4-35. Attribute Definitions - Rename Objects*

| Attribute | Interpretation |
|---|---|
| UUIDName | Used to identify the Advant object by its unique name (UUIDName) and the category of the basic property (EONameCategory), where the name is in. |
| EONameCategory | |
| EOName | The new name value. |
| AspectDescription | Description of the Basic Property aspect keeping the object name. |
| EOIk | System internal identifier, generated by the Export function |

# Chapter 5 Maintenance

## 5.1 Error Messages

This section contains all the messages, which can appear in the log-file.
They are divided in 4 categories: Info, Warning, Error and Sever Error, and sorted alphabetically.

*Table 5-1. Error Messages in the Import Log File*

| Message Texts, Causes, and Actions |
|---|
| **`Info: Create object <name>`**<br>Cause: The message is printed during import/create data for each object, which is created. |
| **`Info: Change I-O allocation`**<br>Cause: The message is printed during re-import if the location in the OCS structure of an existing object differs from the data file definition. The location of the object will be changed then. |
| **`Info: Delete Aspect`**<br>Cause: Is printed every time an aspect is deleted. |
| **`Info: Insert Aspect`**<br>Cause: Is printed every time an aspect is created. |
| **`Info: Duplicate Aspect`**<br>Cause: Is printed every time an aspect is duplicated. |
| **`Info: Line <no> Instantiate <aspect> <EO> from <Lib-EO>`**<br>Cause: The information is printed for each line in the customer data file. It displays the line number in the file in <no>, the object to be instantiated in <EO> and the library object in <Lib-EO>. <aspect> is printed only during import/instantiate aspects. It is omitted when import/instantiate objects. |
| **`Info: processing Line <no>`**<br>Cause: The information is printed for each line in the customer data file when import/create data. |
| **`Info: Update Aspect`**<br>Cause: Is printed every time an aspect is updated. |
| **`Info: Override aspects of <object> using library object <library object>`**<br>Cause: The message is printed during import in Override mode, if the library reference of an existing object is changed. |
| **`Warning: Attribute <attr> is not in the data file`**<br>Cause: The required attribute in <attr> is not in the data file. |

*Table 5-1. Error Messages in the Import Log File (Continued)*

| Message Texts, Causes, and Actions |
|---|
| **`Warning: invalid method <method>`**<br>Cause: There are invalid entries in the used mapping file. |
| **`Warning: Line <no> Instance object is not defined`**<br>Cause: The target object name is not defined in the import data file. The line is skipped and import continues with the next object. |
| **`Warning: No parent object defined in structure <struc-name>`**<br>Cause: The import data file contains an empty field where a name to identify a structure object is expected. No structure aspect is instantiated in structure <stuc-name>. |
| **`Warning: Object <name> not found in Project <Proj-name>`**<br>Cause: An object <name> specified as library reference could not be found in the project <Proj-name>. <line-no> is the line number in the import data file where the error occurred. This info line is normally added to a previous error line.<br>Actions: Check the import data file. The specified library object may be misspelled.<br>Check that the object is in the project data base. |
| **`Warning: Parent object <p-name> not found in structure <struc-name>`**<br>Cause: It was tried to place an engineering object into structure <struc-name> below an object <p-name>, which is not in this structure. |
| **`Warning: token <string> not substituted`**<br>**`Warning: token <string> not found in mapping file`**<br>Cause: This messages can rise if a selected mapping file doesn’t fit to the specified import data file. The import result in this case is undefined. Also a invalid installation can be the reason for it.<br>Actions: Check the data file/mapping file combination in the *Import window* and check that the data file fits to the defined mapping. |
| **`Error: AspectType is not defined`**<br>Cause: No value is entered in an attribute, which requires an aspect type. |
| **`Error: Could not open customer data file <file>`**<br>Cause: The file access to <file> failed. Reason can be an invalid file specification in the *Import window* or a too many open file situation.<br>Actions: Check that the file is available or restart the structure builder. |
| **`Error: Could not open mapping file`**<br>Cause: The specified mapping file could not be found. Reason can be an invalid file specification in the *Import window* or a too many open file situation.<br>Actions: Check that the mapping file is available or restart the structure builder. |
| **`Error: Cross Reference List overflow`**<br>Cause: Software error. |

*Table 5-1. Error Messages in the Import Log File (Continued)*

| Message Texts, Causes, and Actions |
|---|
| **`Error: File access on <template-file>`**<br>Cause: This error an rise in the prepare phase if a file access to an internal *.tpl file failed. Reason can be an invalid installation or a too many open file situation.<br>Actions: Check if <template-file> is available or restart the structure builder. |
| **`Error: Import cancelled by user request`**<br>Cause: Import was cancelled.<br>Actions: Check the import data file. The specified parent object may be misspelled. Check that the object is in the project data base. |
| **`Error: Import is already running`**<br>Cause: If one application starts the import function a 2.nd time this messages is displayed in the log-file. |
| **`Error: in column <no>, <reason>`**<br>Cause: Import data file for import/create contains invalid data, which could be detected. <no> contains the column no, reason is an error text. |
| **`Error: internal line <no> at token <string>`**<br>Cause: Software error or invalid installation template files *.tpl or syntax error in template files *.tpl. |
| **`Error: invalid AspectType <type> in system <system>`**<br>Cause: A specified aspect type <type> for an aspect system <system> in an import data file is not valid. |
| **`Error: invalid EONameCategory <value>`**<br>Cause: A specified attribute `EONameCategory` in an import data file is not valid. |
| **`Error: Invalid Code_* in mapping file - <file>`**<br>Cause: The mapping file <file> content is invalid. It defines an invalid include file. |
| **`Error: Library aspect not found <object name> <aspect identifier>`**<br>Cause: During import/instantiate of aspects the library aspect was not found. <object name> contains the object name, <aspect identifier> either the aspect description or the aspect rel. ref. designation. |
| **`Error: Line <no> Instance object is not defined`**<br>Cause: The target object name is not defined in the import data file. The line is skipped and import continues with the next object. |
| **`Error: Line <no> Library object is not defined`**<br>Cause: The source object name is not defined in the import data file. The line is skipped and import continues with the next object. |
| **`Error: memory problem in expression tree - <node-type>`**<br>Cause: Software error. |

*Table 5-1. Error Messages in the Import Log File (Continued)*

| Message Texts, Causes, and Actions |
|---|
| **`Error: no exec code available`**<br>Cause: Software error. |
| **`Error: Object <name> not found`**<br>Cause: During import/instantiate of aspects the object was not found. <name> contains the object name. |
| **`Error: Object <name> not found in Project <project>`**<br>Cause: During import/instantiate of aspects the library object was not found. <name> contains the library object name, <project> the source project name. |
| **`Error: too many/few identifier - use either description or relrefdes`**<br>Cause: Can rise at import/instantiate of aspects only. The aspect identification can be done either by the aspect description or the aspect relrefdes. In both attributes (LibAspDesc and LibAspRelRef) are values entered.<br>Actions: Remove the values in one of the attributes (LibAspDesc or LibAspRelRef). |
| **`Error: Token buffer overflow`**<br>Cause: Invalid installation or error in template files *.tpl. |
| **`Error: YACC internal - Bad node type`**<br>Cause: Software error. |
| **`Error: [<no>] in line <line-no>`**<br>Cause: This is an universal error line defining the internal error in <no> and the customer data file line number which caused the error in <line-no>. It is followed by one or more additional info lines giving detail informations. |
| **`Sever Error: Invalid Code_* in mapping-file - <file> not found`**<br>Cause: The mapping file content is invalid. |
| **`Sever Error: Invalid mapping-file - too little columns in line <line>`**<br>Cause: The mapping file content is invalid. <line> has less attributes as the 1.st line. |
| **`Sever Error: Invalid mapping-file - missing attribute <attribute>`**<br>Cause: The mapping file content is invalid. Attribute <attribute> is not in the 1.st line. |
| **`Sever Error: Line <line no> - Invalid delimiter <delimiter>`**<br>Cause: The data file, which should be imported, contains another delimiter token as defined in the *Import Data window.*<br>Actions: Check used delimiter token in the data file and setup the same delimiter *token in the Import Data window.* |

# 5.2 Recommendations

## 5.2.1 File Format of Import Data Files

Import data files are ASCII files with *.csv format. They have to fulfill following rules:

- The 1st line in the file has to contain the attribute names. Take care for correct spelling.
- All other lines in the file contain values, related to the attributes. The number of values in one line has to fit to the number of attributes in the 1st line.
- Empty attributes in a data line are allowed.
- Empty lines in a data file are allowed, but not in the 1st line.
- The single attributes and values has to be separated by a delimiter token. The delimiter token must not change in one data file. The comma (,) character is the default delimiter. Other delimiter can also be used.
- If a value contains a delimiter, the value has to be enclosed in double quotation marks.
- Double quotation marks in a quoted string have to be doubled.
- *.**csv** as file extension is recommended but not required.

**Example: Preparing import data with Microsoft Excel.**

- In the first row enter the corresponding attribute names (depends on the kind of import, see for example).
- Format the cells as Text.
- Enter your data.
- If the data cells covers special characters like: - , = and so on, insert a **single quotation mark** in front of the special character (example: **’= Function Structure**).
- Save the file in *.csv format.

The example in Figure 5-1 below shows the content of an import data file, which can be used as for file to **Instantiate Station Objects** with mapping file **objects.csv**.

```
Name,Description,LibRef1,LibRef2,OCSParent,Designation,Bus,Station,Type
Stn_1T,Test Station 1,AC70,,Node-1,ST-1,,3,CI810
Stn_2T,Test Station 2,AC70,,”Node,1”,ST-2,2,4,CI810
```

*Figure 5-1. Example Import Data File*

# Appendix A Overviews, Samples

## A.1 Overview - Instantiate Behavior of Structure Aspects

*Table A-1. Import/Instantiate Behavior of Structure Aspects.*

| **Mapping File** | **Function Structure** | **OCS Structure** | **Product Structure** | **MLocation Structure** | **Attribute(s) for Structure Location Identification 1)** | **Attribute for Designation Values 2)** | **Sub-structure Instantiate and Naming Rule 3)** |
|---|---|---|---|---|---|---|---|
| signal.csv | X | | | | FuncParent | | |
| | | X | | | Node,<br>**Designation**,<br>**Channel** **4)** | | |
| board.csv | | X | | | OCSParent | OCSDesig | 2 |
| station.csv | | X | | | OCSParent | Designation | |
| objects.csv | X | | | | FuncParent | | 2 |
| | | X | | | OCSParent | OCSDesig | |
| caee_obj.csv<br>caee_obj_ard.csv | X | | | | FuncParent | FuncDesig | 3 |
| | | X | | | OCSParent | OCSDesig | 3 |
| | | | X | | ProdParent | ProdDesig | 3 |
| | | | | X | MLocParent | MLocDesig | 3 |
| caee_brd.csv<br>caee_brd_ard.csv | | X | | | OCSParent | OCSDesig | 3 |
| | | | X | | ProdParent | ProdDesig | 3 |
| | | | | X | MLocParent,<br>**Rack**,<br>**Slot** | | 3 |

*Table A-1. Import/Instantiate Behavior of Structure Aspects.*

| **Mapping File** | **Function Structure** | **OCS Structure** | **Product Structure** | **MLocation Structure** | **Attribute(s) for Structure Location Identification 1)** | **Attribute for Designation Values 2)** | **Sub-structure Instantiate and Naming Rule 3)** |
|---|---|---|---|---|---|---|---|
| caee_sig.csv<br>caee_sig_ard.csv | X | | | | FuncParent | FuncDesig | |
| | | X | | | Node, **4)**<br>**Designation**,<br>**Channel** | | |

**1)** Attribute name(s), which keep the information for identifying the parent object in the structure.
Normally the parent object name has to be filled in.
Use relative reference designations for **bold attributes**.

**2)** The content of the attribute is used as the Rel.Ref.Des. value for the related structure aspect.
It is set when the structure aspect is instantiated. During a re-import it is not updated.

**3)** If marked, sub-structure objects, as available in the library, are created during the 1st import.
During a re-import it is not updated (the location in the structure is not changed).
For sub-structure objects two naming rules are available:

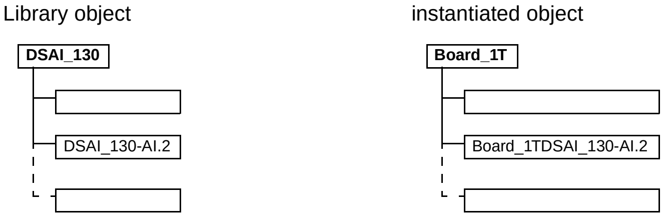
**Naming rule 1**
The name of the created object is composed of the name of the instantiated object and the name of the library sub-structure object.

**Naming rule 2**
The name of the created object is composed of the name of the instantiated object and the name of the library sub-structure object.
If library object name than is identical with the first part of this created name, this part is cut off.

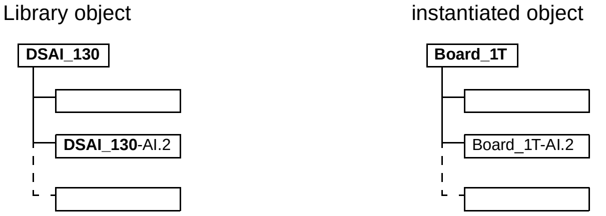
**Naming rule 3**
The name of the created object is composed of the name of the instantiated object and the name of the library sub-structure object.
Wherever the library object name than is found in this created name, it is replaced by the name of the instantiated object.

Examples:

Example - naming rule 1



Example - naming rule 2



Example - naming rule 3

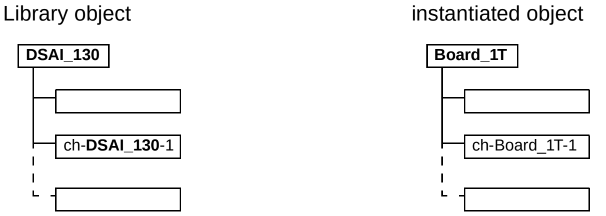


*Figure A-1. Examples - Naming Rules*

**4)** Is updated during a re-import (the location is changed according to the definition in the import data file).

## A.2 Sample - Instantiate Signal Objects

This sample shows what happens when instantiate signal objects, and how the data in the import data file have to refer to the library reference and to Structure Parent Objects.

Starting with a data file in Table A-2, containing the information to instantiate one signal object, the result is shown in Table A-3.

This file has to be imported with mapping **signals.csv**.

```
Name,Description,LibRef1,LibRef2,FuncParent,Node,Designation,Channel
Sig1,Sample-Signal,DIS,,M09,Node-1,DI5,1
```

*Figure A-2. Signal Data File*

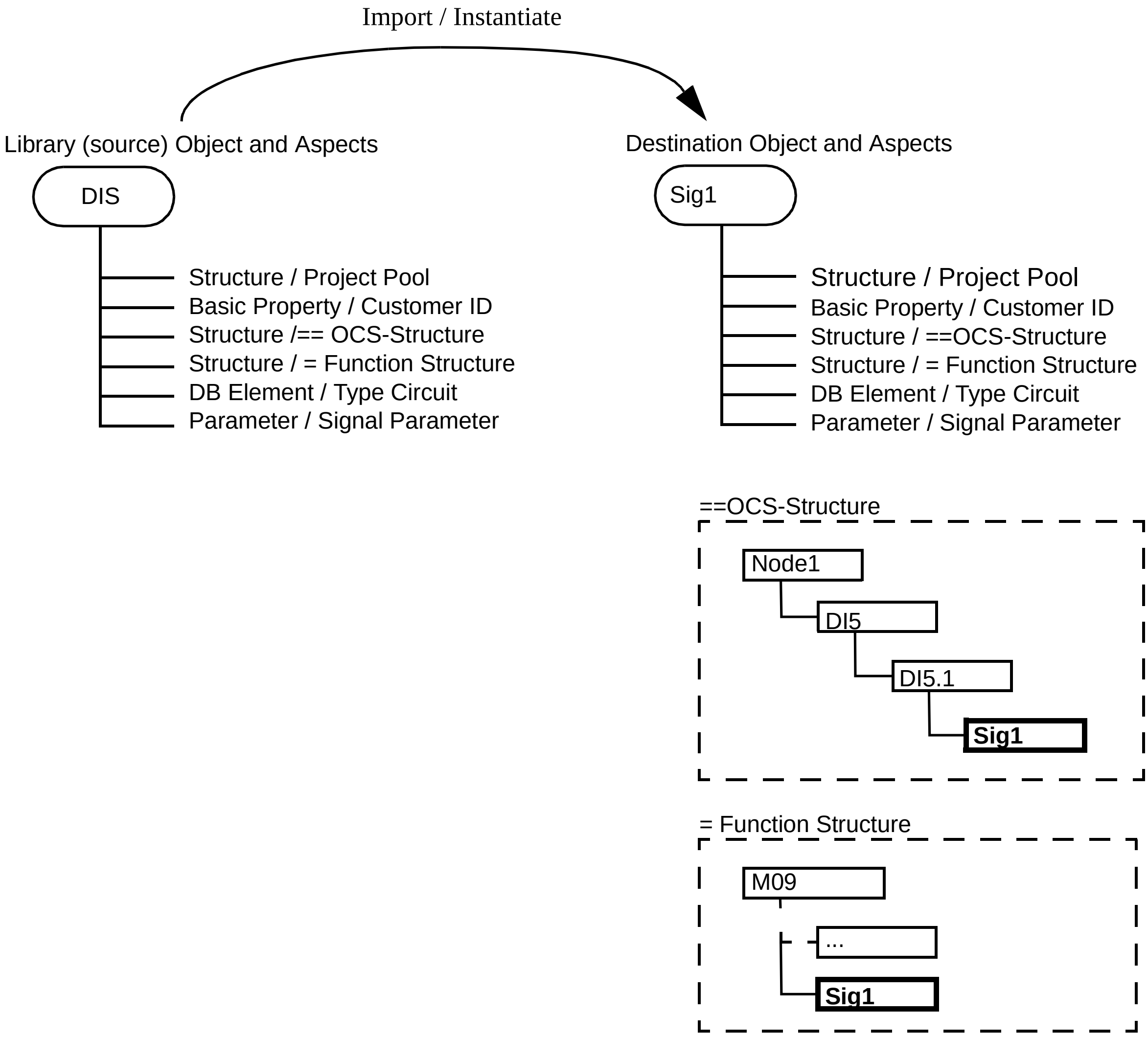

Import / Instantiate
Library (source) Object and Aspects
DIS
Structure / Project Pool
Basic Property / Customer ID
Structure /== OCS-Structure
Structure / = Function Structure
DB Element / Type Circuit
Parameter / Signal Parameter
Destination Object and Aspects
Sig1
Structure / Project Pool
Basic Property / Customer ID
Structure / ==OCS-Structure
Structure / = Function Structure
DB Element / Type Circuit
Parameter / Signal Parameter
==OCS-Structure
Node1
DI5
DI5.1
Sig1
= Function Structure
M09
...
Sig1

*Figure A-3. Example - Instantiate Signal Object*

# INDEX

**A**
Absolute Reference Designation 2-3, 4-3 to 4-5, 4-8
aspect system 3-6

**B**
Bad file 2-9, 2-11 to 2-13

**C**
Configuration data 2-6, 3-1, 3-5

**D**
Delimiter 2-9, 2-11 to 2-12

**E**
export folder 3-8
export range 3-1
Export.log 3-8
Export.set 3-7

**I**
Identification 2-3 to 2-4

**K**
Keep mode 2-3 to 2-5, 2-9

**L**
Library reference 2-1, A-3
Log file 2-1 to 2-2, 2-8, 2-10, 2-12, 3-4, 3-8

**M**
Mapping files 2-7, 2-12 to 2-13

**N**
Naming rule A-2 to A-3

**O**
Override mode 2-3 to 2-5
Override options 2-4, 2-9, 2-11

**P**
Progress window 3-5

**R**
Relative Reference Designation 2-2 to 2-4

**S**
Set file 3-7, 3-9
Structure Parent Objects 2-2, A-3
sub-structure 2-2 to 2-3, 2-5

**U**
unique names 3-4, 4-25

**W**
working box 3-5

3BDS100581R1

October 1999